AutoCAD 2013

AutoLISP Reference Guide

January 2012

Autodesk®

Contents

AutoLISP Functions 1

AutoLISP Functions

The following is a catalog of the AutoLISP® functions available in AutoCAD®. The functions are listed alphabetically.

In this chapter, each listing contains a brief description of the function's use and a function syntax statement showing the order and the type of arguments required by the function.

Note that any functions, variables, or features not described here or in other parts of the documentation are not officially supported and are subject to change in future releases.

For information on syntax, see AutoLISP Function Syntax in the *AutoLISP Developer's Guide*.

Note that the value returned by some functions is categorized as *unspecified*. This indicates you cannot rely on using the value returned from this function.

Operators

+ (add)

Returns the sum of all numbers.

```
(+
  [number number]
 ...)
```

Arguments

number A number.

Return Values

The result of the addition. If you supply only one *number* argument, this function returns the result of adding it to zero. If you supply no arguments, the function returns 0.

Examples

```
(+ 1 2)
  returns
  3
(+ 1 2 3 4.5)
  returns
  10.5
(+ 1 2 3 4.0)
  returns
  10.0
```

- (subtract)

Subtracts the second and following numbers from the first and returns the difference

```
(-
  [number number]
 ...)
```

Arguments

number A number.

Return Values

The result of the subtraction. If you supply more than two *number* arguments, this function returns the result of subtracting the sum of the second through the last numbers from the first number. If you supply only one *number* argument, this function subtracts the number from zero, and returns a negative number. Supplying no arguments returns 0.

Examples

```
(- 50 40)
  returns
  10
(- 50 40.0)
  returns
  10.0
(- 50 40.0 2.5)
  returns
  7.5
(- 8)
  returns
  -8
```

* (multiply)

Returns the product of all numbers

```
(*
  [number number]
 ...)
```

Arguments

number A number.

Return Values

The result of the multiplication. If you supply only one *number* argument, this function returns the result of multiplying it by one; it returns the number. Supplying no arguments returns 0.

Examples

```
(* 2 3)
  returns
 6
(* 2 3.0)
  returns
 6.0
(* 2 3 4.0)
```

```
  returns
  24.0
(* 3 -4.5)
  returns
  -13.5
(* 3)
  returns
  3
```

/ (divide)

Divides the first number by the product of the remaining numbers and returns the quotient

```
(/
  [number number]
 ...)
```

Arguments

number A number.

Return Values

The result of the division. If you supply more than two *number* arguments, this function divides the first number by the product of the second through the last numbers, and returns the final quotient. If you supply one *number* argument, this function returns the result of dividing it by one; it returns the number. Supplying no arguments returns 0.

Examples

```
(/ 100 2)
  returns
  50
(/ 100 2.0)
  returns
  50.0
(/ 100 20.0 2)
  returns
```

```
  2.5
(/ 100 20 2)
  returns
  2
(/ 4)
  returns
  4
```

= (equal to)

Compares arguments for numerical equality

```
(=
  numstr [numstr]
 ...)
```

Arguments

numstr A number or a string.

Return Values

`T`, if all arguments are numerically equal; otherwise `nil` . If only one argument is supplied, **=** returns `T`.

Examples

```
(= 4 4.0)
  returns
  T
(= 20 388)
  returns
  nil
(= 2.4 2.4 2.4)
  returns
  T
(= 499 499 500)
  returns
  nil
(= "me" "me")
  returns
```

```
  T
(= "me" "you")
  returns
  nil
```

See also:

The eq (page 80) and equal (page 81) functions.

/= (not equal to)

Compares arguments for numerical inequality

```
(/=
  numstr [numstr]
 ...)
```

Arguments

numstr A number or a string.

Return Values

T, if no two successive arguments are the same in value; otherwise nil. If only one argument is supplied, /= returns T.

Note that the behavior of /= does not quite conform to other LISP dialects. The standard behavior is to return T if no two arguments in the list have the same value. In AutoLISP, /= returns T if no *successive* arguments have the same value; see the examples that follow.

Examples

```
(/= 10 20)
  returns
  T
(/= "you" "you")
  returns
  nil
(/= 5.43 5.44)
  returns
```

```
  T
(/= 10 20 10 20 20)
  returns
  nil
(/= 10 20 10 20)
  returns
  T
```

NOTE In the last example, although there are two arguments in the list with the same value, they do not follow one another; thus /= evaluates to T.

< (less than)

Returns *T* if each argument is numerically less than the argument to its right; otherwise *nil*

```
(<
  numstr [numstr]
 ...)
```

Arguments

numstr A number or a string.

Return Values

T, if each argument is numerically less than the argument to its right; otherwise returns nil . If only one argument is supplied, < returns T.

Examples

```
(< 10 20)
  returns
  T
(< "b" "c")
  returns
  T
(< 357 33.2)
  returns
  nil
(< 2 3 88)
```

```
  returns
  T
(< 2 3 4 4)
  returns
  nil
```

<= (less than or equal to)

Returns *T* if each argument is numerically less than or equal to the argument to its right; otherwise returns *nil*

```
(<=
  numstr [numstr]
 ...)
```

Arguments

numstr A number or a string.

Return Values

`T`, if each argument is numerically less than or equal to the argument to its right; otherwise returns `nil`. If only one argument is supplied, **<=** returns `T`.

Examples

```
(<= 10 20)
  returns
  T
(<= "b" "b")
  returns
  T
(<= 357 33.2)
  returns
  nil
(<= 2 9 9)
  returns
  T
(<= 2 9 4 5)
  returns
  nil
```

> (greater than)

Returns *T* if each argument is numerically greater than the argument to its right; otherwise returns *nil*

```
(>
  numstr [numstr]
 ...)
```

Arguments

numstr A number or a string.

Return Values

`T`, if each argument is numerically greater than the argument to its right; otherwise `nil`. If only one argument is supplied, > returns `T`.

Examples

```
(> 120 17)
  returns
  T
(> "c" "b")
  returns
  T
(> 3.5 1792)
  returns
  nil
(> 77 4 2)
  returns
  T
(> 77 4 4)
  returns
  nil
```

>= (greater than or equal to)

Returns *T* if each argument is numerically greater than or equal to the argument to its right; otherwise returns *nil*

```
(>=
  numstr [numstr]
 ...)
```

Arguments

numstr A number or a string.

Return Values

T, if each argument is numerically greater than or equal to the argument to its right; otherwise nil. If only one argument is supplied, **>=** returns T.

Examples

```
(>= 120 17)
  returns
  T
(>= "C" "C")
  returns
  T
(>= 3.5 1792)
  returns
  nil
(>= 77 4 4)
  returns
  T
(>= 77 4 9)
  returns
  nil
```

~ (bitwise NOT)

Returns the bitwise NOT (1's complement) of the argument

```
(~
  int
)
```

Arguments

int An integer.

Return Values

The bitwise NOT (1's complement) of the argument.

Examples

```
(~ 3)
  returns
  -4
(~ 100)
  returns
  -101
(~ -4)
  returns
  3
```

1+ (increment)

Increments a number by 1

```
(1+
  number
)
```

Arguments

number Any number.

Return Values

The argument, increased by 1.

Examples

```
(1+ 5)
  returns
  6
(1+ -17.5)
  returns
  -16.5
```

1- (decrement)

Decrements a number by 1

```
(1-
  number
)
```

Arguments

number Any number.

Return Values

The argument, reduced by 1.

Examples

```
(1- 5)
  returns
  4
(1- -17.5)
  returns
  -18.5
```

A Functions

abs

Returns the absolute value of a number

```
(abs
  number
)
```

Arguments

number Any number.

Return Values

The absolute value of the argument.

Examples

```
(abs 100)
  returns
  100
(abs -100)
  returns
  100
(abs -99.25)
  returns
  99.25
```

acad-pop-dbmod

Restores the value of the DBMOD system variable to the value that was most recently stored with acad-push-dbmod

```
(acad-pop-dbmod)
```

This function is used with **acad-push-dbmod** to control the DBMOD system variable. The DBMOD system variable tracks changes to a drawing and triggers save-drawing queries.

This function is implemented in *acapp.arx*, which is loaded by default. This function pops the current value of the DBMOD system variable off an internal stack.

Return Values

Returns T if successful; otherwise, if the stack is empty, returns nil.

acad-push-dbmod

Stores the current value of the DBMOD system variable

```
(acad-push-dbmod)
```

This function is used with **acad-pop-dbmod** to control the DBMOD system variable. You can use this function to change a drawing without changing

the DBMOD system variable. The DBMOD system variable tracks changes to a drawing and triggers save-drawing queries.

This function is implemented in *acapp.arx*, which is loaded by default. This function pushes the current value of the DBMOD system variable onto an internal stack. To use **acad-push-dbmod** and **acad-pop-dbmod**, precede operations with **acad-push-dbmod** and then use **acad-pop-dbmod** to restore the original value of the DBMOD system variable.

Return Values

Always returns T.

Examples

The following example shows how to store the modification status of a drawing, change the status, and then restore the original status.

```
(acad-push-dbmod)
(setq new_line '((0 . "LINE") (100 . "AcDbEntity") (8 .
"0")
             (100 . "AcDbLine") (10 1.0 2.0 0.0) (11 2.0
1.0 0.0)
             (210 0.0 0.0 1.0)))
(entmake new_line)            ; Set DBMOD to flag 1
(command "_color" "2")        ; Set DBMOD to flag 4
(command "_-vports" "_SI")    ; Set DBMOD to flag 8
(command "_vpoint" "0,0,1")   ; Set DBMOD to flag 16
(acad-pop-dbmod)             ; Set DBMOD to original value
```

acad_strlsort

Sorts a list of strings in alphabetical order

```
(acad_strlsort
  list
)
```

Arguments

list The list of strings to be sorted.

Return Values

The *list* in alphabetical order. If the list is invalid or if there is not enough memory to do the sort, **acad_strlsort** returns `nil`.

Examples

Sort a list of abbreviated month names:

```
Command: (setq mos '("Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug"
"Sep" "Oct" "Nov" "Dec"))
("Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct"
"Nov" "Dec")
Command: (acad_strlsort mos)
("Apr" "Aug" "Dec" "Feb" "Jan" "Jul" "Jun" "Mar" "May" "Nov"
"Oct" "Sep")
```

acad_truecolorcli

Prompts for colors at the command line

```
(acad_truecolorcli
  color [allowbylayer] [alternatePrompt]
)
```

Arguments

color A dotted pair that describes the default color. The first element of the dotted pair must be one of the color-related DXF group codes (62, 420, or 430); for example, `(62 . ColorIndex)`, `(420 . TrueColor)`, or `(430 . "colorbook$colorname")`.

allowbylayer Omitting the *allowbylayer* argument or setting it to a non-`nil` value enables entering bylayer or byblock to set the color. If set to `nil`, an error results if bylayer or byblock is entered.

alternateprompt An optional prompt string. If this string is omitted, the default value is “New color”.

Return Values

When the operation is successful, the function returns a list of one or more dotted pairs (depending on the tab on which the color is selected) describing the color selected. The last dotted pair in the list indicates the color selected. The function returns `nil` if the user cancels the function.

Color book color If the last item in the returned list is a 430 pair, then the specified color originates from a color book. This returned list will also contain

a 420 pair that describes the corresponding true color and a 62 pair that describes the closest matching color index value.

True color If the returned list contains a 420 pair as the last item, then a true color was specified (as "Red,Green,Blue"). The list will also contain a 62 pair that indicates the closest matching color index. No 430 pair will be present.

Color index If the last item in the list is a 62 pair, then a colorindex was chosen. No other dotted pairs will be present in the returned list.

Examples

Prompt for a color selection at the command line with a purple color index default selection and alternative text for the command prompt:

```
Command: (acad_truecolorcli '(62 . 215) 1 "Pick a color")
New Color [Truecolor/COlorbook] <215>:
((62 . 215))
```

Prompt for a color selection at the command line with a yellow color index default selection, then set the color by layer:

```
Command: (acad_truecolorcli '(62 . 2))
New Color [Truecolor/COlorbook] <2 (yellow)>: bylayer
((62 . 256))
```

acad_truecolordlg

Displays the AutoCAD color selection dialog box with tabs for index color, true color, and color books

```
(acad_truecolordlg
  color [allowbylayer] [currentlayercolor]
)
```

Arguments

color A dotted pair that describes the default color. The first element of the dotted pair must be one of the color-related DXF group codes (62, 420, or 430); for example, `(62 . ColorIndex)`, `(420 . TrueColor)`, or `(430 . "colorbook$colorname")`.

allowbylayer If set to `nil`, disables the ByLayer and ByBlock buttons. Omitting the *allowbylayer* argument or setting it to a non-`nil` value enables the ByLayer and ByBlock buttons.

currentlayercolor Optional dotted pair in the same form as `color` that sets the value of the bylayer/byblock color in the dialog.

Return Values

When the operation is successful, the function returns a list of one or more dotted pairs (depending on the tab on which the color is selected) describing the color selected. The last dotted pair in the list indicates the color selected. The function returns `nil` if the user cancels the dialog box.

Color book color If the last item in the returned list is a 430 pair, then the specified color originates from a color book. This returned list will also contain a 420 pair that describes the corresponding true color and a 62 pair that describes the closest matching color index value.

True color If the returned list contains a 420 pair as the last item, then a true color was specified (as "Red,Green,Blue"). The list will also contain a 62 pair that indicates the closest matching color index. No 430 pair will be present.

Color index If the last item in the list is a 62 pair, then a color index was chosen. No other dotted pairs will be present in the returned list.

Examples

Open the color selection dialog to the Color Index tab and accept the purple default selection:

```
Command: (acad_truecolordlg '(62 . 215))
((62 . 215))
```

Open the color selection dialog to the True Color tab with a green default selection and with the By Layer and By Block buttons disabled:

```
Command: (acad_truecolordlg '(420 . 2686760) nil)
((62 . 80) (420 . 2686760))
```

Open the color selection dialog to the Color Books tab and accept the mustard default selection:

```
Command: (acad_truecolordlg '(430 . "RAL CLASSIC$RAL 1003"))
((62 . 40) (420 . 16235019) (430 . "RAL CLASSIC$RAL 1003"))
```

acdimenableupdate

Controls the automatic updating of associative dimensions

```
(acdimenableupdate nil | T)
```

The **acdimenableupdate** function is intended for developers who are editing geometry and don't want the dimension to be updated until after the edits are complete.

Arguments

nil Associative dimensions will not update (even if the geometry is modified) until the DIMREGEN command is entered.

T Enable automatic updating of associative dimensions when the geometry is modified.

Return Values

`nil`

Examples

Disable the automatic update of associative dimensions in the drawing:
`Command:` **(acdimenableupdate nil)**

Enable the automatic update of associative dimensions in the drawing:
`Command:` **(acdimenableupdate T)**

acet-layerp-mode

Queries and sets the LAYERPMODE setting

```
(acet-layerp-mode [
  status
])
```

Arguments

status Specifying `T` turns LAYERPMODE on, enabling layer-change tracking. `Nil` turns LAYERPMODE off.

If this argument is not present, **acet-layerp-mode** returns the current status of LAYERPMODE.

Return Values

`T` if current status of LAYERPMODE is on; `nil` if LAYERPMODE is off.

Examples

Check the current status of LAYERPMODE:
`Command:` **(acet-layerp-mode)**

```
T
```

Turn LAYERPMODE off:

```
Command: (acet-layerp-mode nil)
nil
```

Check the current status of LAYERPMODE:

```
Command: (acet-layerp-mode)
nil
```

See also:

The LAYERP and LAYERPMODE commands in the *Command Reference*.

acet-layerp-mark

Places beginning and ending marks for Layer Previous recording

```
(acet-layerp-mark [
  status
])
```

The **acet-layerp-mark** function allows you to group multiple layer commands into a single transaction so that they can be undone by issuing LAYERP a single time. LAYERPMODE must be on in order to set marks.

Arguments

status Specifying `T` sets a begin mark. Specifying `nil` sets an end mark, clearing the begin mark.

If *status* is omitted, `acet-layerp-mark` returns the current mark status for layer settings.

Return Values

`T` if a begin mark is in effect; otherwise `nil`.

Examples

The following code changes layer 0 to blue, and then makes several additional layer changes between a set of begin and end marks. If you issue LAYERP after running this code, layer 0 reverts to blue.

```
(defun TestLayerP ()
```

```
  ;; Turn LAYERPMODE on, if it isn't already
  (if (not (acet-layerp-mode))
    (acet-layerp-mode T)
  )
  ;; Set layer 0 to the color blue
  (command "_.layer" "_color" "blue" "0" "")
  ;; Set a begin mark
  (acet-layerp-mark T)
  ;; Issue a series of layer commands, and then set an end
 mark
  (command "_.layer" "_color" "green" "0" "")
  (command "_.layer" "_thaw" "*" "")
  (command "_.layer" "_unlock" "*" "")
  (command "_.layer" "_ltype" "hidden" "0" "")
  (command "_.layer" "_color" "red" "0" "")
  ;; Set an end mark
  (acet-layerp-mark nil)
)
```

See also:

The LAYERP command in the *Command Reference*.

alert

Displays a dialog box containing an error or warning message

```
(alert
  string
)
```

Arguments

string The string to appear in the alert box.

Return Values

`nil`

Examples

Display a message in an alert box:

```
(alert "That function is not available.")
```

Display a multiple line message, by using the newline character in *string*:

```
(alert "That function\nis not available.")
```

NOTE

Line length and the number of lines in an alert box are platform, device, and window dependent. AutoCAD truncates any string that is too long to fit inside an alert box.

alloc

Sets the size of the segment to be used by the *expand* function

```
(alloc n-alloc)
```

Arguments

n-alloc An integer indicating the amount of memory to be allocated. The integer represents the number of symbols, strings, usubrs, reals, and cons cells.

Return Values

The previous setting of `n-alloc`.

Examples

```
_$
  (alloc 100)

1000
```

See also:

The expand (page 85) function.

and

Returns the logical AND of the supplied arguments

```
(and
```

```
    [expr
   ...
    ]
  )
```

Arguments

expr Any expression.

Return Values

`Nil`, if any of the expressions evaluate to `nil; otherwise T`. If **and** is issued without arguments, it returns `T`.

Examples

```
Command: (setq a 103 b nil c "string")
"string"
Command: (and 1.4 a c)
T
Command: (and 1.4 a b c)
nil
```

angle

Returns an angle in radians of a line defined by two endpoints

```
(angle
  pt1 pt2
)
```

Arguments

pt1 An endpoint.

pt2 An endpoint.

Return Values

An angle, in radians.

The angle is measured from the *X* axis of the current construction plane, in radians, with angles increasing in the counterclockwise direction. If 3D points are supplied, they are projected onto the current construction plane.

Examples

```
Command: (angle '(1.0 1.0) '(1.0 4.0))
1.5708
Command: (angle '(5.0 1.33) '(2.4 1.33))
3.14159
```

See also:

The topic in the Angular Conversion *AutoLISP Developer's Guide*.

angtof

Converts a string representing an angle into a real (floating-point) value in radians

```
(angtof
   string [units]
)
```

Arguments

string A string describing an angle based on the format specified by the *mode* argument. The *string* must be a string that **angtof** can parse correctly to the specified *unit*. It can be in the same form that **angtos** returns, or in a form that AutoCAD allows for keyboard entry.

units Specifies the units in which the string is formatted. The value should correspond to values allowed for the AutoCAD system variable AUNITS in the *Command Reference*. If *unit* is omitted, **angtof** uses the current value of AUNITS. The following *unit*s may be specified:

0 -- Degrees

1 -- Degrees/minutes/seconds

2 -- Grads

3 -- Radians

4 -- Surveyor's units

Return Values

A real value, if successful; otherwise `nil`.

The **angtof** and **angtos** functions are complementary: if you pass **angtof** a string created by **angtos**, **angtof** is guaranteed to return a valid value, and vice versa (assuming the *unit* values match).

Examples

```
Command: (angtof "45.0000")
0.785398
Command: (angtof "45.0000" 3)
1.0177
```

See also:

The angtos (page 24) function.

angtos

Converts an angular value in radians into a string

```
(angtos
  angle [unit [precision]]
)
```

Arguments

angle A real number, in radians.

unit An integer that specifies the angular units. If *unit* is omitted, **angtos** uses the current value of the AutoCAD system variable AUNITS. The following *unit*s may be specified:

0 -- Degrees

1 -- Degrees/minutes/seconds

2 -- Grads

3 -- Radians

4 -- Surveyor's units

precision An integer specifying the number of decimal places of precision to be returned. If omitted, **angtos** uses the current setting of the AutoCAD system variable AUPREC in the *Command Reference*.

The **angtos** function takes *angle* and returns it edited into a string according to the settings of *unit*, *precision*, the AutoCAD UNITMODE system variable, and the DIMZIN dimensioning variable in the *Command Reference*.

The **angtos** function accepts a negative *angle* argument, but always reduces it to a positive value between zero and 2 pi radians before performing the specified conversion.

The UNITMODE system variable affects the returned string when surveyor's units are selected (a *unit* value of 4). If UNITMODE = 0, spaces are included in the string (for example, "N 45d E"); if UNITMODE = 1, no spaces are included in the string (for example, "N45dE").

Return Values

A string, if successful; otherwise `nil`.

Examples

```
Command: (angtos 0.785398 0 4)
"45.0000"
Command: (angtos -0.785398 0 4)
"315.0000"
Command: (angtos -0.785398 4)
"S 45d E"
```

NOTE Routines that use the **angtos** function to display arbitrary angles (those not relative to the value of ANGBASE) should check and consider the value of ANGBASE.

See also:

The angtof (page 23) function and String Conversions in the *AutoLISP Developer's Guide*.

append

Takes any number of lists and appends them together as one list

```
(append
  [list
 ...
  ]
)
```

Arguments

list A list.

Return Values

A list with all arguments appended to the original. If no arguments are supplied, **append** returns `nil`.

Examples

```
Command: (append '(a b) '(c d))
(A B C D)
Command: (append '((a)(b)) '((c)(d)))
((A) (B) (C) (D))
```

apply

Passes a list of arguments to, and executes, a specified function

```
(apply '
  function list
)
```

Arguments

'function A function. The *function* argument can be either a symbol identifying a **defun**, or a **lambda** expression.

list A list. Can be `nil`, if the function accepts no arguments.

Return Values

The result of the function call.

Examples

```
Command: (apply '+ '(1 2 3))
6
Command: (apply 'strcat '("a" "b" "c"))
"abc"
```

arx

Returns a list of the currently loaded ObjectARX applications

```
(arx)
```

Return Values

A list of ObjectARX® application file names; the path is not included in the file name.

Examples

```
Command: (arx)
("layermanager.bundle" "mtextformat.bundle" "opm.bundle")
```

See also:

The arxload (page 27) and arxunload (page 28) functions.

arxload

Loads an ObjectARX application

```
(arxload
  application [onfailure]
)
```

Arguments

application A quoted string or a variable that contains the name of an executable file. You can omit the *.bundle* extension from the file name.

You must supply the full path name of the ObjectARX executable file, unless the file is in a directory that is in the AutoCAD support file search path.

onfailure An expression to be executed if the load fails.

Return Values

The application name, if successful. If unsuccessful and the *onfailure* argument is supplied, **arxload** returns the value of this argument; otherwise, failure results in an error message.

If you attempt to load an application that is already loaded, **arxload** issues an error message. You may want to check the currently loaded ObjectARX applications with the **arx** function before using **arxload**.

Examples

Load the *acbrowser.bundle* file supplied in the AutoCAD installation directory:

```
Command: (arxload "/Applications/Autodesk/AutoCAD
2013/AutoCAD 2013.app/acbrowser.bundle")
"/Applications/Autodesk/AutoCAD 2013/AutoCAD
2013.app/acbrowser.bundle"
```

See also:

The arxunload (page 28) function.

arxunload

Unloads an ObjectARX application

```
(arxunload
  application [onfailure]
)
```

Arguments

application A quoted string or a variable that contains the name of a file that was loaded with the **arxload** function. You can omit the *.bundle* extension and the path from the file name.

onfailure An expression to be executed if the unload fails.

Return Values

The application name, if successful. If unsuccessful and the *onfailure* argument is supplied, **arxunload** returns the value of this argument; otherwise, failure results in an error message.

Note that locked ObjectARX applications cannot be unloaded. ObjectARX applications are locked by default.

Examples

Unload the acbrowse application that was loaded in the arxload function example:

```
Command: (arxunload "acbrowser")
"acbrowser"
```

See also:

The arxload (page 27) function.

ascii

Returns the conversion of the first character of a string into its ASCII character code (an integer)

```
(ascii
  string
```

```
)
```

Arguments

string A string.

Return Values

An integer.

Examples

```
Command: (ascii "A")
65
Command: (ascii "a")
97
Command: (ascii "BIG")
66
```

assoc

Searches an association list for an element and returns that association list entry

```
(assoc
  element alist
)
```

Arguments

element Key of an element in an association list.

alist An association list to be searched.

Return Values

The *alist* entry, if successful. If **assoc** does not find *element* as a key in *alist*, it returns `nil`.

Examples

```
Command: (setq al '((name box) (width 3) (size 4.7263) (depth 5)))
((NAME BOX) (WIDTH 3) (SIZE 4.7263) (DEPTH 5))
Command: (assoc 'size al)
(SIZE 4.7263)
Command: (assoc 'weight al)
nil
```

atan

Returns the arctangent of a number in radians

```
(atan
  num1 [num2]
)
```

Arguments

num1 A number.

num2 A number.

Return Values

The arctangent of *num1*, in radians, if only *num1* is supplied. If you supply both *num1* and *num2* arguments, **atan** returns the arctangent of *num1*/*num2*, in radians. If *num2* is zero, it returns an angle of plus or minus 1.570796 radians (+90 degrees or -90 degrees), depending on the sign of*num1*. The range of angles returned is -pi/2 to +pi/2 radians.

Examples

```
Command: (atan 1)
0.785398
Command: (atan 1.0)
0.785398
Command: (atan 0.5)
0.463648
Command: (atan 1.0)
0.785398
Command: (atan -1.0)
-0.785398
Command: (atan 2.0 3.0)
0.588003
Command: (atan 2.0 -3.0)
2.55359
Command: (atan 1.0 0.0)
1.5708
```

atof

Converts a string into a real number

```
(atof
  string
)
```

Arguments

string A string to be converted into a real number.

Return Values

A real number.

Examples

```
Command: (atof "97.1")
97.1
Command: (atof "3")
3.0
Command: (atof "3.9")
3.9
```

atoi

Converts a string into an integer

```
(atoi
  string
)
```

Arguments

string A string to be converted into an integer.

Return Values

An integer.

Examples

```
Command: (atoi "97")
97
Command: (atoi "3")
```

```
3
Command: (atoi "3.9")
3
```

See also:

The itoa (page 125) function.

atom

Verifies that an item is an atom

```
(atom
  item
)
```

Arguments

item Any AutoLISP element.

Some versions of LISP differ in their interpretation of **atom**, so be careful when converting from non-AutoLISP code.

Return Values

`Nil` if *item* is a list; otherwise `T`. Anything that is not a list is considered an atom.

Examples

```
Command: (setq a '(x y z))
(X Y Z)
Command: (setq b 'a)
A
Command: (atom 'a)
T
Command: (atom a)
nil
Command: (atom 'b)
T
Command: (atom b)
T
Command: (atom '(a b c))
nil
```

atoms-family

Returns a list of the currently defined symbols

```
(atoms-family
  format [symlist]
)
```

Arguments

format An integer value of 0 or 1 that determines the format in which **atoms-family** returns the symbol names:

0 Return the symbol names as a list

1 Return the symbol names as a list of strings

symlist A list of strings that specify the symbol names you want **atoms-family** to search for.

Return Values

A list of symbols. If you specify **symlist**, then **atoms-family** returns the specified symbols that are currently defined, and returns `nil` for those symbols that are not defined.

Examples

```
Command: (atoms-family 0)
(BNS_PRE_SEL FITSTR2LEN C:AI_SPHERE ALERT DEFUN C:BEXTEND
REM_GROUP
B_RESTORE_SYSVARS BNS_CMD_EXIT LISPED FNSPLITL...
```

The following code verifies that the symbols `CAR`, `CDR`, and `XYZ` are defined, and returns the list as strings:

```
Command: (atoms-family 1 '("CAR" "CDR" "XYZ"))
("CAR" "CDR" nil)
```

The return value shows that the symbol `XYZ` is not defined.

autoarxload

Predefines command names to load an associated ObjectARX file

```
(autoarxload
```

```
    filename cmdlist
)
```

The first time a user enters a command specified in *cmdlist*, AutoCAD loads the ObjectARX application specified in *filename*, then continues the command.

If you associate a command with *filename* and that command is not defined in the specified file, AutoCAD alerts you with an error message when you enter the command.

Arguments

filename A string specifying the *.bundle* file to be loaded when one of the commands defined by the *cmdlist* argument is entered at the Command prompt. If you omit the path from *filename*, AutoCAD looks for the file in the support file search path.

cmdlist A list of strings.

Return Values

```
nil
```

Examples

The following code defines the `C:APP1`, `C:APP2`, and `C:APP3` functions to load the *bonusapp.bundle* file:

```
(autoarxload "BONUSAPP" '("APP1" "APP2" "APP3"))
```

autoload

Predefines command names to load an associated AutoLISP file

```
(autoload
  filename cmdlist
)
```

The first time a user enters a command specified in *cmdlist*, AutoCAD loads the application specified in *filename*, then continues the command.

Arguments

filename A string specifying the *.lsp* file to be loaded when one of the commands defined by the *cmdlist* argument is entered at the Command

prompt. If you omit the path from *filename*, AutoCAD looks for the file in the Support File Search Path.

cmdlist A list of strings.

Return Values

```
nil
```

If you associate a command with *filename* and that command is not defined in the specified file, AutoCAD alerts you with an error message when you enter the command.

Examples

The following causes AutoCAD to load the *bonusapp.lsp* file the first time the APP1, APP2, or APP3 commands are entered at the Command prompt:

```
(autoload "BONUSAPP" '("APP1" "APP2" "APP3"))
```

B Functions

Boole

Serves as a general bitwise Boolean function

```
(Boole
  operator int1 [int2
...
  ]
)
```

Arguments

operator An integer between 0 and 15 representing one of the 16 possible Boolean functions in two variables.

int1, int2... Integers.

Note that **Boole** will accept a single integer argument, but the result is unpredictable.

Successive integer arguments are bitwise (logically) combined based on this function and on the following truth table:

Boolean truth table

Int1	Int2	operator bit
0	0	8
0	1	4
1	0	2
1	1	1

Each bit of *int1* is paired with the corresponding bit of *int2*, specifying one horizontal row of the truth table. The resulting bit is either 0 or 1, depending on the setting of the *operator* bit that corresponds to this row of the truth table.

If the appropriate bit is set in *operator*, the resulting bit is 1; otherwise the resulting bit is 0. Some of the values for *operator* are equivalent to the standard Boolean operations AND, OR, XOR, and NOR.

Boole function bit values

Operator	Operation	Resulting bit is 1 if
1	AND	Both input bits are 1
6	XOR	Only one of the two input bits is 1
7	OR	Either or both of the input bits are 1
8	NOR	Both input bits are 0 (1's complement)

Return Values

An integer.

Examples

The following specifies a logical AND of the values 12 and 5:

```
Command: (Boole 1 12 5)
4
```

The following specifies a logical XOR of the values 6 and 5:

```
Command: (Boole 6 6 5)
3
```

You can use other values of *operator* to perform other Boolean operations for which there are no standard names. For example, if *operator* is 4, the resulting bits are set if the corresponding bits are set in *int2* but not in *int1*:

```
Command: (Boole 4 3 14)
12
```

boundp

Verifies if a value is bound to a symbol

```
(boundp
  sym
)
```

Arguments

sym A symbol.

Return Values

`T` if *sym* has a value bound to it. If no value is bound to *sym*, or if it has been bound to `nil`, **boundp** returns `nil`. If *sym* is an undefined symbol, it is automatically created and is bound to `nil`.

Examples

```
Command: (setq a 2 b nil)
nil
Command: (boundp 'a)
T
Command: (boundp 'b)
nil
```

The **atoms-family** function provides an alternative method of determining the existence of a symbol without automatically creating the symbol.

See also:

The atoms-family (page 33) function.

C Functions

caddr

Returns the third element of a list

```
(caddr
  list
)
```

In AutoLISP, **caddr** is frequently used to obtain the *Z* coordinate of a 3D point (the third element of a list of three reals).

Arguments

list A list.

Return Values

The third element in *list;* otherwise `nil`, if the list is empty or contains fewer than three elements.

Examples

```
Command: (setq pt3 '(5.25 1.0 3.0))
(5.25 1.0 3.0)
Command: (caddr pt3)
3.0
Command: (caddr '(5.25 1.0))
nil
```

See also:

The Point Lists topic in the *AutoLISP Developer's Guide*.

cadr

Returns the second element of a list

```
(cadr
  list
)
```

In AutoLISP, **cadr** is frequently used to obtain the *Y* coordinate of a 2D or 3D point (the second element of a list of two or three reals).

Arguments

list A list.

Return Values

The second element in *list;* otherwise `nil`, if the list is empty or contains only one element.

Examples

```
Command: (setq pt2 '(5.25 1.0))
(5.25 1.0)
Command: (cadr pt2)
1.0
Command: (cadr '(4.0))
nil
Command: (cadr '(5.25 1.0 3.0))
1.0
```

See also:

The Point Lists topic in the *AutoLISP Developer's Guide*.

car

Returns the first element of a list

```
(car
  list
)
```

Arguments

list A list.

Return Values

The first element in *list;* otherwise `nil`, if the list is empty.

Examples

```
Command: (car '(a b c))
A
Command: (car '((a b) c))
(A B)
Command: (car '())
nil
```

See also:

The Point Lists topic in the *AutoLISP Developer's Guide*.

cdr

Returns a list containing all but the first element of the specified list

```
(cdr
  list
)
```

Arguments

list A list.

Return Values

A list containing all the elements of *list,* except the first element (but see Note below). If the list is empty, **cdr** returns `nil`.

NOTE When the *list* argument is a dotted pair, **cdr** returns the second element without enclosing it in a list.

Examples

```
Command: (cdr '(a b c))
(B C)
Command: (cdr '((a b) c))
(C)
Command: (cdr '())
nil
Command: (cdr '(a . b))
B
```

```
Command: (cdr '(1 . "Text"))
"Text"
```

See also:

The Point Lists topic in the *AutoLISP Developer's Guide*.

chr

Converts an integer representing an ASCII character code into a single-character string

```
(chr
  integer
)
```

Arguments

list An integer.

Return Values

A string containing the ASCII character code for *integer*. If the integer is not in the range of 1-255, the return value is unpredictable.

Examples

```
Command: (chr 65)
"A"
Command: (chr 66)
"B"
Command: (chr 97)
"a"
```

close

Closes an open file

```
(close
  file-desc
)
```

Arguments

file-desc A file descriptor obtained from the **open** function.

Return Values

`Nil` if *file-desc* is valid; otherwise results in an error message.

After a **close**, the file descriptor is unchanged but is no longer valid. Data added to an open file is not actually written until the file is closed.

Examples

The following code counts the number of lines in the file *somefile.txt* and sets the variable `ct` equal to that number:

```
(setq fil "SOMEFILE.TXT")
(setq x (open fil "r") ct 0)
(while (read-line x)
  (setq ct (1+ ct))
)
(close x)
```

command

Executes an AutoCAD command

```
(command
  [arguments]
 ...)
```

Arguments

arguments AutoCAD commands and their options.

The *arguments* to the **command** function can be strings, reals, integers, or points, as expected by the prompt sequence of the executed command. A null string (`""`) is equivalent to pressing Enter on the keyboard. Invoking **command** with no argument is equivalent to pressing Esc and cancels most AutoCAD commands.

The **command** function evaluates each argument and sends it to AutoCAD in response to successive prompts. It submits command names and options as strings, 2D points as lists of two reals, and 3D points as lists of three reals. AutoCAD recognizes command names only when it issues a Command prompt.

Return Values

```
nil
```

Examples

The following example sets two variables `pt1` and `pt2` equal to two point values 1,1 and 1,5. It then uses the **command** function to issue the LINE command in the *Command Reference* and pass the two point values.

```
Command: (setq pt1 '(1 1) pt2 '(1 5))
(1 5)
Command: (command "line" pt1 pt2 "")
line From point:
To point:
To point:
Command: nil
```

Restrictions and Notes

Also, if you use the **command** function in an *acad.lsp* or *.mnl* file, it should be called only from within a **defun** statement. Use the **S::STARTUP** function to define commands that need to be issued immediately when you begin a drawing session.

For AutoCAD commands that require the selection of an object (like the BREAK and TRIM commands in the *Command Reference*), you can supply a list obtained with **entsel** instead of a point to select the object. For examples, see Passing Pick Points yo AutoCAD Commands in the *AutoLISP Developer's Guide*.

Commands executed from the **command** function are not echoed to the command line if the CMDECHO system variable (accessible from **setvar** and **getvar**) is set to 0.

NOTE When using the SCRIPT command with the **command** function, it should be the last function call in the AutoLISP routine.

See also:

initcommandversion (page 116)

vl-cmdf (page 219) under Command Submission in the *AutoLISP Developer's Guide*

command-s

Executes an AutoCAD command and the supplied input.

```
(command-s
  cmdname [arguments]
)
```

Arguments

cmdname Name of the command to execute.

arguments The command input to supply to the command being executed.

The arguments to the command function can be strings, reals, integers, or points, as expected by the prompt sequence of the executed command. A null string ("") is equivalent to pressing Enter on the keyboard.

Return Values

`nil` is returned by the function when the command is done executing on the provided arguments. An `*error*` is returned when the function fails to complete successfully.

Examples

The following example demonstrates how to execute the CIRCLE command and create a circle with a diameter of 2.75.

```
Command: (command-s "_circle" "5,4" "_d" 2.75)
nil
```

The following example demonstrates how to prompt the user for the center point of the circle.

```
Command: (setq cPt (getpoint "\nSpecify center point: "))
(5.0 4.0 0.0)

Command: (command-s "_circle" cPt "_d" 2.75)
nil
```

The following is an invalid use of prompting for user input with the command-s function.

```
Command: (command-s "_circle" (getpoint "\nSpecify center point:
 ") "_d" 2.75)
```

Differences from the `Command` Function

The `command-s` function is a variation of the `command` function which has some restrictions on command token content, but is both faster than command and can be used in `*error*` handlers due to internal logic differences.

A command token is a single argument provided to the command-s function. This could be a string, real, integer, point, entity name, list, and so on. The following example shows the LINE command and three command tokens:

```
(command-s "_line" "0,0" "5,7" "")
```

The "-s" suffix stands for "subroutine" execution of the supplied command tokens. In this form, AutoCAD is directly called from AutoLISP, processes the supplied command tokens in a temporary command processor distinct from the main document command processor, and then returns, thus terminating the temporary command processor. The command that is being executed must be started and completed in the same command-s function.

In contrast, the command function remains a "co-routine" execution of the supplied command tokens, where AutoLISP evaluates the tokens one at a time, sending the result to AutoCAD, and then returning to allow AutoCAD to process that token. AutoCAD then calls AutoLISP back, and AutoLISP resumes evaluation of the expression in progress. In this logic flow, subsequent token expressions can query AutoCAD for the results of previous token processing and use it.

In summary, the "co-routine" style of command token processing is more functionally powerful, but is limited in when it can be used when running. The "subroutine" style of command token processing can be used in a much wider range of contexts, but processes all command tokens in advance, and actual execution is non-interactive. For the same set of command tokens, command-s function is significantly faster.

Known Considerations

When using the command-s function, you must take the following into consideration:

- Token streams fed in a single command-s expression must represent a full command and its input. Any commands in progress when command tokens are all processed will be cancelled. The following is not valid with the command-s function:

  ```
  (command-s "_line")
  (command-s "2,2" "12.25,9" "")
  ```

- All command tokens will be evaluated before they are handed over to AutoCAD for execution. In contrast, the command function actually performs each command token evaluation and then feeds the result to AutoCAD, which processes it before the next command token is processed.

- No "Pause" command tokens may be used. Expressions that interact with the drawing area or Command Window may be used, but will all be processed before AutoCAD receives and processes any of them.

  The following is not valid with the `command-s` function:

```
(command-s "_line" "0,0" PAUSE "")
```

IMPORTANT Although the `command-s` function is similar to the `command` function, caution should be taken when using U or UNDO to roll back the system state if there is an AutoCAD command already in progress when the AutoLISP expression is entered. In that case, the results of running UNDO may cause the command in progress to fail or even crash AutoCAD.

`*error*` Handler

If your *error* handler uses the `command` function, consider updating the way you define your custom `*error*` handlers using the following methods:

- **Substitute `command-s` for `command` in `*error*` handler**

  For typical `*error*` handler cases where the previous state of the program needs to be restored and a few batch commands are executed, you can substitute `(command-s <...>)` for `(command <...>)`. The `*error*` handler is called from the same context as it always has been.

  The following demonstrates a based `*error*` handler using the `command-s` function:

```
(defun my_err(s)
  (prompt "\nERROR: mycmd failed or was cancelled")
  (setvar "clayer" old_clayer)
  (command-s "_.UNDO" "_E")
  (setq *error* mv_oer)
)

(defun c:mycmd ()
  (setq old_err *error*
        *error* my_err
        old_clayer (getvar "clayer")
  )

  (setq insPt (getpoint "\nSpecify text insertion: "))

  (if (/= insPt nil)
    (progn
```

```
        (command-s "_.UNDO" "_BE")
        (command-s "-_LAYER" "_M" "Text" "_C" "3" "" "")
        (command-s "_-TEXT" insPt "" "0" "Sample Text")
        (command-s "_.UNDO" "_E")
      )
    )

    (setvar "clayer" old_clayer)
    (setq *error* mv_oer)
   (princ)
  )
```

- **Retaining the use of the `command` function in `*error*` handler**

  If using the `command-s` function is not viable option, then the `command` function can still be used, but only at the expense of losing access to any local symbols that would normally be on the AutoLISP call stack at the time of the `*error*` processing.

  The following is an overview of what is required to continue to use the `command` function in the `*error*` handler.

  - When overriding the `*error*` symbol with a custom `*error*` handler, invoke the `*push-error-using-command*` function to inform AutoLISP that error handling will be used with the proceeding `command` functions.

    NOTE Whenever an AutoLISP expression evaluation begins, the AutoLISP engine assumes that the `command` function will not be allowed within an `*error*` handler.

  - If the `*error*` handler refers to local symbols that are on the AutoLISP stack at the point where AutoLISP program failed or was cancelled, you must remove those references, or make the referenced symbols global symbols.

    All local symbols on the AutoLISP call stack are pushed out of scope because the AutoLISP evaluator is reset before entering the `*error*` handler.

  Now the `command` function can be used within the `*error*` handler.

  However, if your program actually pushes and pops error handlers as part of its operations, or your AutoLISP logic can be invoked while other

unknown AutoLISP logic is invoked, there are a couple more steps you may have to make.

- When restoring an old error handler, also invoke the *pop-error-mode* function to reverse the effects of any call to the *push-error-using-command* or *push-error-using-stack* functions.
- If your logic has nested pushes and pops of the *error* handler, and an *error* handler has been set up to use the command function by invoking *push-error-using-command*, while the nested handler will not use it, you can provide access to the locally defined symbols on the AutoLISP stack by invoking *push-error-using-stack* at the same point where you set *error* to the current handler. If this is done, you must also invoke *pop-error-mode* after the old *error* handler is restored.

See also:

Command (page 42)

cond

Serves as the primary conditional function for AutoLISP

```
(cond
  [
(
  test result
 ...) ...
  ]
)
```

The **cond** function accepts any number of lists as arguments. It evaluates the first item in each list (in the order supplied) until one of these items returns a value other than nil. It then evaluates those expressions that follow the test that succeeded.

Return Values

The value of the last expression in the sublist. If there is only one expression in the sublist (that is, if *result* is missing), the value of the *test* expression is returned. If no arguments are supplied, **cond** returns nil.

Examples

The following example uses **cond** to perform an absolute value calculation:

```
(cond
   ((minusp a) (- a))
   (t a)
)
```

If the variable `a` is set to the value-10, this returns 10.

As shown, **cond** can be used as a *case* type function. It is common to use `T` as the last (default) *test* expression. Here's another simple example. Given a user response string in the variable `s`, this function tests the response and returns 1 if it is `Y` or `y`, 0 if it is `N` or `n`; otherwise `nil`.

```
(cond
   ((= s "Y") 1)
   ((= s "y") 1)
   ((= s "N") 0)
   ((= s "n") 0)
   (t nil)
)
```

cons

Adds an element to the beginning of a list, or constructs a dotted list

```
(cons
  new-first-element list-or-atom
)
```

Arguments

new-first-element Element to be added to the beginning of a list. This element can be an atom or a list.

list-or-atom A list or an atom.

Return Values

The value returned depends on the data type of *list-or-atom*. If *list-or-atom* is a list, **cons** returns that list with *new-first-element* added as the first item in the

list. If *list-or-atom* is an atom, **cons** returns a dotted pair consisting of *new-first-element* and *list-or-atom*.

Examples

```
Command: (cons 'a '(b c d))
(A B C D)
Command: (cons '(a) '(b c d))
((A) B C D)
Command: (cons 'a 2)
(A . 2)
```

See also:

The List Handling topic in the *AutoLISP Developer's Guide*.

cos

Returns the cosine of an angle expressed in radians

```
(cos
  ang
)
```

Arguments

ang An angle, in radians.

Return Values

The cosine of *ang*, in radians.

Examples

```
Command: (cos 0.0)
1.0
Command: (cos pi)
-1.0
```

cvunit

Converts a value from one unit of measurement to another

```
(cvunit
   value from-unit to-unit
)
```

Arguments

value The numeric value or point list (2D or 3D point) to be converted.

from-unit The unit that *value* is being converted from.

to-unit The unit that *value* is being converted to.

The *from-unit* and *to-unit* arguments can name any unit type found in the *acad.unt* file.

Return Values

The converted value, if successful; otherwise `nil`, if either unit name is unknown (not found in the *acad.unt* file), or if the two units are incompatible (for example, trying to convert grams into years).

Examples

```
Command: (cvunit 1 "minute" "second")
60.0
Command: (cvunit 1 "gallon" "furlong")
nil
Command: (cvunit 1.0 "inch" "cm")
2.54
Command: (cvunit 1.0 "acre" "sq yard")
4840.0
Command: (cvunit '(1.0 2.5) "ft" "in")
(12.0 30.0)
Command: (cvunit '(1 2 3) "ft" "in")
(12.0 24.0 36.0)
```

NOTE

If you have several values to convert in the same manner, it is more efficient to convert the value 1.0 once and then apply the resulting value as a scale factor in your own function or computation. This works for all predefined units except temperature, where an offset is involved as well.

See also:

The Unit Conversion topic in the *AutoLISP Developer's Guide*.

D Functions

defun

Defines a function

```
(defun
  sym ([arguments] [/ variables...]
) expr...)
```

Arguments

sym A symbol naming the function.

arguments The names of arguments expected by the function.

/ variables The names of one or more local variables for the function.

The slash preceding the variable names must be separated from the first local name and from the last argument, if any, by at least one space.

expr Any number of AutoLISP expressions to be evaluated when the function executes.

If you do not declare any arguments or local symbols, you must supply an empty set of parentheses after the function name.

If duplicate argument or symbol names are specified, AutoLISP uses the first occurrence of each name and ignores the following occurrences.

Return Values

The result of the last expression evaluated.

WARNING Never use the name of a built-in function or symbol for the *sym* argument to **defun**. This overwrites the original definition and makes the built-in function or symbol inaccessible. To get a list of built-in and previously defined functions, use the **atoms-family** function.

Examples

```
(defun myfunc (x y) ...)
  Function takes two arguments
```

```
(defun myfunc (/ a b) ...)
  Function has two local variables

(defun myfunc (x / temp) ...)
  One argument, one local variable

(defun myfunc () ...)
  No arguments or local variables
```

See also:

The Symbol and Function Handling topic in the *AutoLISP Developer's Guide*.

defun-q

Defines a function as a list

```
(defun-q
  sym ([arguments] [/ variables...]
) expr...)
```

The **defun-q** function is provided strictly for backward-compatibility with previous versions of AutoLISP, and should not be used for other purposes. You can use **defun-q** in situations where you need to access a function definition as a list structure, which is the way **defun** was implemented in previous, non-compiled versions of AutoLISP.

Arguments

sym A symbol naming the function.

arguments The names of arguments expected by the function.

/ variables The names of one or more local variables for the function.

The slash preceding the variable names must be separated from the first local name and from the last argument, if any, by at least one space.

expr Any number of AutoLISP expressions to be evaluated when the function executes.

If you do not declare any arguments or local symbols, you must supply an empty set of parentheses after the function name.

If duplicate argument or symbol names are specified, AutoLISP uses the first occurrence of each name and ignores the following occurrences.

Return Values

The result of the last expression evaluated.

Examples

(defun-q my-startup (x) (print (list x)))

```
MY-STARTUP
```

(my-startup 5)

```
(5) (5)
```

Use **defun-q-list-ref** to display the list structure of **my-startup**:

(defun-q-list-ref 'my-startup)

```
((X) (PRINT (LIST X)))
```

See also:

The defun-q-list-ref (page 54) and defun-q-list-set (page 55) functions.

defun-q-list-ref

Displays the list structure of a function defined with defun-q

```
(defun-q-list-ref '
  function
)
```

Arguments

function A symbol naming the function.

Return Values

The list definition of the function; otherwise `nil`, if the argument is not a list.

Examples

Define a function using **defun-q**:

(defun-q my-startup (x) (print (list x)))

```
MY-STARTUP
```

Use **defun-q-list-ref** to display the list structure of **my-startup**:

```
(defun-q-list-ref 'my-startup)
((X) (PRINT (LIST X)))
```

See also:

The defun-q (page 53) and defun-q-list-set (page 55) functions.

defun-q-list-set

Sets the value of a symbol to be a function defined by a list

```
(defun-q-list-set '
  sym list
)
```

Arguments

sym A symbol naming the function

list A list containing the expressions to be included in the function.

Return Values

The *sym* defined.

Examples

```
(defun-q-list-set 'foo '((x) x))
FOO
(foo 3)
3
```

The following example illustrates the use of **defun-q-list-set** to combine two functions into a single function. First, from the Visual LISP Console window, define two functions with **defun-q**:

```
(defun-q s::startup (x) (print x))
S::STARTUP
(defun-q my-startup (x) (print (list x)))
MY-STARTUP
```

Use **defun-q-list-set** to combine the functions into a single function:

```
(defun-q-list-set 's::startup (append
  (defun-q-list-ref 's::startup)
  (cdr (defun-q-list-ref 'my-startup))))
S::STARTUP
```

The following illustrates how the functions respond individually, and how the functions work after being combined using **defun-q-list-set**:

```
(defun-q foo (x) (print (list 'foo x)))
FOO
(foo 1)
(FOO 1) (FOO 1)
(defun-q bar (x) (print (list 'bar x)))
BAR
(bar 2)
(BAR 2) (BAR 2)
(defun-q-list-set
  'foo
  (append (defun-q-list-ref 'foo)
          (cdr (defun-q-list-ref 'bar))
  ))
FOO
(foo 3)
(FOO 3) (BAR 3) (BAR 3)
```

See also:

The defun-q (page 53) and defun-q-list-ref (page 54) functions.

dictadd

Adds a nongraphical object to the specified dictionary

```
(dictadd
  ename symbol newobj
)
```

Arguments

ename Name of the dictionary the object is being added to.

symbol The key name of the object being added to the dictionary; *symbol* must be a unique name that does not already exist in the dictionary.

newobj A nongraphical object to be added to the dictionary.

As a general rule, each object added to a dictionary must be unique to that dictionary. This is specifically a problem when adding group objects to the group dictionary. Adding the same group object using different key names results in duplicate group names, which can send the **dictnext** function into an infinite loop.

Return Values

The entity name of the object added to the dictionary.

Examples

The examples that follow create objects and add them to the named object dictionary.

Create a dictionary entry list:

```
Command: (setq dictionary (list '(0 . "DICTIONARY") '(100 . "AcDbDictionary")))
((0 . "DICTIONARY") (100 . "AcDbDictionary"))
```

Create a dictionary object using the **entmakex** function:

```
Command: (setq xname (entmakex dictionary))
<Entity name: 1d98950>
```

Add the dictionary to the named object dictionary:

```
Command: (setq newdict (dictadd (namedobjdict) "MY_WAY_COOL_DICTIONARY" xname))
<Entity name: 1d98950>
```

Create an Xrecord list:

```
Command: (setq datalist (append (list '(0 . "XRECORD")'(100 . "AcDbXrecord")) '((1 . "This is my data") (10 1. 2. 3.) (70 . 33))))
((0 . "XRECORD") (100 . "AcDbXrecord") (1 . "This is my data") (10 1.0 2.0 3.0) (70 . 33))
```

Make an Xrecord object:

```
Command: (setq xname (entmakex datalist))
<Entity name: 1d98958>
```

Add the Xrecord object to the dictionary:

```
Command: (dictadd newdict "DATA_RECORD_1" xname)
```

```
<Entity name: 1d98958>
```

See also:

The dictnext (page 58), dictremove (page 59), dictrename (page 60), dictsearch (page 61), and namedobjdict (page 149) functions.

dictnext

Finds the next item in a dictionary

```
(dictnext
  ename [rewind]
)
```

Arguments

ename Name of the dictionary being viewed.

rewind If this argument is present and is not `nil`, the dictionary is rewound and the first entry in it is retrieved.

Return Values

The next entry in the specified dictionary; otherwise `nil`, when the end of the dictionary is reached. Entries are returned as lists of dotted pairs of DXF-type codes and values. Deleted dictionary entries are not returned.

The **dictsearch** function specifies the initial entry retrieved.

Use **namedobjdict** to obtain the master dictionary entity name.

NOTE Once you begin stepping through the contents of a dictionary, passing a different dictionary name to **dictnext** will cause the place to be lost in the original dictionary. In other words, only one global iterator is maintained for use in this function.

Examples

Create a dictionary and an entry as shown in the example for **dictadd**. Then make another Xrecord object:

```
Command: (setq xname (entmakex datalist))
<Entity name: 1b62d60>
```

Add this Xrecord object to the dictionary, as the second record in the dictionary:

```
Command: (dictadd newdict "DATA_RECORD_2" xname)
<Entity name: 1b62d60>
```

Return the entity name of the next entry in the dictionary:

```
Command: (cdr (car (dictnext newdict)))
<Entity name: 1bac958>
```

dictnext returns the name of the first entity added to the dictionary.

Return the entity name of the next entry in the dictionary:

```
Command: (cdr (car (dictnext newdict)))
<Entity name: 1bac960>
```

dictnext returns the name of the second entity added to the dictionary.

Return the entity name of the next entry in the dictionary:

```
Command: (cdr (car (dictnext newdict)))
nil
```

There are no more entries in the dictionary, so **dictnext** returns `nil`.

Rewind to the first entry in the dictionary and return the entity name of that entry:

```
Command: (cdr (car (dictnext newdict T)))
<Entity name: 1bac958>
```

Specifying `T` for the optional *rewind* argument causes **dictnext** to return the first entry in the dictionary.

See also:

The dictadd (page 56), dictremove (page 59), dictrename (page 60), dictsearch (page 61), and namedobjdict (page 149) functions.

dictremove

Removes an entry from the specified dictionary

```
(dictremove
  ename symbol
)
```

By default, removing an entry from a dictionary does not delete it from the database. This must be done with a call to **entdel**. Currently, the exceptions to this rule are groups and mlinestyles. The code that implements these features requires that the database and these dictionaries be up to date and, therefore, automatically deletes the entity when it is removed (with **dictremove**) from the dictionary.

Arguments

ename Name of the dictionary being modified.

symbol The entry to be removed from *ename*.

The **dictremove** function does not allow the removal of an mlinestyle from the mlinestyle dictionary if it is actively referenced by an mline in the database.

Return Values

The entity name of the removed entry. If *ename* is invalid or *symbol* is not found, **dictremove** returns `nil`.

Examples

The following example removes the dictionary created in the **dictadd** example:

```
Command: (dictremove (namedobjdict) "my_way_cool_dictionary")
<Entity name: 1d98950>
```

See also:

The dictadd (page 56), dictnext (page 58), dictrename (page 60), dictsearch (page 61), and namedobjdict (page 149) functions.

dictrename

Renames a dictionary entry

```
(dictrename
  ename oldsym newsym
)
```

Arguments

ename Name of the dictionary being modified.

oldsym Original key name of the entry.

newsym New key name of the entry.

Return Values

The *newsym* value, if the rename is successful. If the *oldname* is not present in the dictionary, or if *ename* or *newname* is invalid, or if *newname* is already present in the dictionary, then **dictrename** returns `nil`.

Examples

The following example renames the dictionary created in the **dictadd** sample:

```
Command: (dictrename (namedobjdict) "my_way_cool_dictionary"
"An even cooler dictionary")
"An even cooler dictionary"
```

See also:

The dictadd (page 56), dictnext (page 58), dictremove (page 59), dictsearch (page 61), and namedobjdict (page 149) functions.

dictsearch

Searches a dictionary for an item

```
(dictsearch
  ename symbol [setnext]
)
```

Arguments

ename Name of the dictionary being searched.

symbol A string that specifies the item to be searched for within the dictionary.

setnext If present and not `nil`, the **dictnext** entry counter is adjusted so the following **dictnext** call returns the entry after the one returned by this **dictsearch** call.

Return Values

The entry for the specified item, if successful; otherwise `nil`, if no entry is found.

Examples

The following example illustrates the use of **dictsearch** to obtain the dictionary added in the **dictadd** example:

```
Command: (setq newdictlist (dictsearch (namedobjdict)
"my_way_cool_dictionary"))
((-1 . <Entity name: 1d98950>) (0 . "DICTIONARY") (5 . "52")
(102 . "{ACAD_REACTORS") (330 . <Entity name: 1d98860>) (102
. "}") (330 . <Entity name: 1d98860>) (100 . "AcDbDictionary")
(280 . 0) (281 . 1) (3 . "DATA_RECORD_1") (350 . <Entity
name: 1d98958>))
```

See also:

The dictadd (page 56), dictnext (page 58), dictremove (page 59), and namedobjdict (page 149) functions.

distance

Returns the 3D distance between two points

```
(distance
  pt1 pt2
)
```

Arguments

pt1 A 2D or 3D point list.

pt1 A 2D or 3D point list.

Return Values

The distance.

If one or both of the supplied points is a 2D point, then **distance** ignores the *Z* coordinates of any 3D points supplied and returns the 2D distance between the points as projected into the current construction plane.

Examples

```
Command: (distance '(1.0 2.5 3.0) '(7.7 2.5 3.0))
6.7
Command: (distance '(1.0 2.0 0.5) '(3.0 4.0 0.5))
2.82843
```

See also:

The Geometric Utilities topic in the *AutoLISP Developer's Guide*.

distof

Converts a string that represents a real (floating-point) value into a real value

```
(distof
   string [mode]
)
```

The **distof** and **rtos** functions are complementary. If you pass **distof** a string created by **rtos**, **distof** is guaranteed to return a valid value, and vice versa (assuming the mode values are the same).

Arguments

string A string to be converted. The argument must be a string that **distof** can parse correctly according to the units specified by *mode*. It can be in the same form that **rtos** returns, or in a form that AutoCAD allows for keyboard entry.

mode The units in which the string is currently formatted. The *mode* corresponds to the values allowed for the AutoCAD system variable LUNITS in the *Command Reference*. Specify one of the following numbers for *mode*:

1 Scientific

2 Decimal

3 Engineering (feet and decimal inches)

4 Architectural (feet and fractional inches)

5 Fractional

Return Values

A real number, if successful; otherwise `nil`.

NOTE The **distof** function treats modes 3 and 4 the same. That is, if *mode* specifies 3 (engineering) or 4 (architectural) units, and *string* is in either of these formats, **distof** returns the correct real value.

---

dumpallproperties

Retrieves an entity's supported properties.

```
(dumpallproperties
  ename [context]
)
```

Arguments

ename Name of the entity being queried. The ename can refer to either a graphical or non-graphical entity.

context Value expected is 0 or 1, the default is 0 when a value is not provided. When 1 is provided as the context, some property values such as Position, Normal, and StartPoint are promoted from a single value to individual X, Y, and Z values.

For example, the following displays the StartPoint first as not being promoted and then being as promoted:

- Not promoted, context = 0

```
StartPoint (type: AcGePoint3d)  (LocalName: StartPoint)
 = 6.250000   8.750000  0.000000
```

- Promoted, context = 1

```
StartPoint/X (type: double)  (LocalName: Start X) =
6.250000
StartPoint/Y (type: double)  (LocalName: Start Y) =
8.750000
StartPoint/Z (type: double)  (LocalName: Start Z) =
0.000000
```

Return Values

`nil` is returned by the function while the properties and their current values are output to the Command Window.

Examples

The following example demonstrates how to list the available properties for a line object with the properties Delta, EndPoint, Normal, and StartPoint promoted to individual values.

```
Command: (setq e1 (car (entsel "\nSelect a line: ")))
Select a line:
```

```
<Entity name: 10e2e4b20>
Command: (dumpAllProperties e1 1)

Begin dumping object (class: AcDbLine)
Angle (type: double)  (RO)  (LocalName: Angle) = 5.159347
Annotative (type: bool)  (LocalName: Annotative) = Failed
 to get value to get value
Area (type: double)  (RO)  (LocalName: Area) = 0.000000
BlockId (type: AcDbObjectId)  (RO) = Ix
CastShadows (type: bool) = 0
ClassName (type: AcString)  (RO) =
Closed (type: bool)  (RO)  (LocalName: Closed) = Failed to
 get value
CollisionType (type: AcDb::CollisionType)  (RO) = 1
Color (type: AcCmColor)  (LocalName: Color) = BYLAYER
Delta/X (type: double)  (RO)  (LocalName: Delta X) =
3.028287
Delta/Y (type: double)  (RO)  (LocalName: Delta Y) =
-6.318026
Delta/Z (type: double)  (RO)  (LocalName: Delta Z) =
0.000000
EndParam (type: double)  (RO) = 7.006281
EndPoint/X (type: double)  (LocalName: End X) = 23.249243
EndPoint/Y (type: double)  (LocalName: End Y) = 11.968958
EndPoint/Z (type: double)  (LocalName: End Z) = 0.000000
ExtensionDictionary (type: AcDbObjectId)  (RO) = Ix
Handle (type: AcDbHandle)  (RO) = 1b2
HasFields (type: bool)  (RO) = 0
HasSaveVersionOverride (type: bool) = 0
Hyperlinks (type: AcDbHyperlink*)
IsA (type: AcRxClass*)  (RO) = AcDbLine
IsAProxy (type: bool)  (RO) = 0
IsCancelling (type: bool)  (RO) = 0
IsEraseStatusToggled (type: bool)  (RO) = 0
IsErased (type: bool)  (RO) = 0
IsModified (type: bool)  (RO) = 0
IsModifiedGraphics (type: bool)  (RO) = 0
IsModifiedXData (type: bool)  (RO) = 0
IsNewObject (type: bool)  (RO) = 0
IsNotifyEnabled (type: bool)  (RO) = 0
IsNotifying (type: bool)  (RO) = 0
IsObjectIdsInFlux (type: bool)  (RO) = 0
IsPeriodic (type: bool)  (RO) = 0
```

```
IsPersistent (type: bool)  (RO) = 1
IsPlanar (type: bool)  (RO) = 1
IsReadEnabled (type: bool)  (RO) = 1
IsReallyClosing (type: bool)  (RO) = 1
IsTransactionResident (type: bool)  (RO) = 0
IsUndoing (type: bool)  (RO) = 0
IsWriteEnabled (type: bool)  (RO) = 0
LayerId (type: AcDbObjectId)  (LocalName: Layer) = Ix
Length (type: double)  (RO)  (LocalName: Length) = 7.006281
LineWeight (type: AcDb::LineWeight)  (LocalName: Lineweight)
 = -1
LinetypeId (type: AcDbObjectId)  (LocalName: Linetype) =
Ix
LinetypeScale (type: double)  (LocalName: Linetype scale)
 = 1.000000
LocalizedName (type: AcString)  (RO) = Line
MaterialId (type: AcDbObjectId)  (LocalName: Material) =
Ix
MergeStyle (type: AcDb::DuplicateRecordCloning)  (RO) = 1
Normal/X (type: double) = 0.000000
Normal/Y (type: double) = 0.000000
Normal/Z (type: double) = 1.000000
ObjectId (type: AcDbObjectId)  (RO) = Ix
OwnerId (type: AcDbObjectId)  (RO) = Ix
PlotStyleName (type: AcString)  (LocalName: Plot style) =
 ByLayer
ReceiveShadows (type: bool) = 0 Failed to get value
StartParam (type: double)  (RO) = 0.000000
StartPoint/X (type: double)  (LocalName: Start X) =
20.220956
StartPoint/Y (type: double)  (LocalName: Start Y) =
18.286984
StartPoint/Z (type: double)  (LocalName: Start Z) = 0.000000
Thickness (type: double)  (LocalName: Thickness) = 0.000000
Transparency (type: AcCmTransparency)  (LocalName:
Transparency) = 0
Visible (type: AcDb::Visibility) = 0
End object dump
```

See also:

GetPropertyValue (page 103)
IsPropertyReadOnly (page 123)

SetPropertyValue (page 175)

E Functions

entdel

Deletes objects (entities) or restores previously deleted objects

```
(entdel
  ename
)
```

The entity specified by *ename* is deleted if it is currently in the drawing. The **entdel** function restores the entity to the drawing if it has been deleted previously in this editing session. Deleted entities are purged from the drawing when the drawing is exited. The **entdel** function can delete both graphical and nongraphical entities.

Arguments

ename Name of the entity to be deleted or restored.

Return Values

The entity name.

Usage Notes

The **entdel** function operates only on main entities. Attributes and polyline vertices cannot be deleted independently of their parent entities. You can use the **command** function to operate the ATTEDIT or PEDIT command in the *Command Reference* to modify subentities.

You cannot delete entities within a block definition. However, you can completely redefine a block definition, minus the entity you want deleted, with **entmake**.

Examples

Get the name of the first entity in the drawing and assign it to variable `e1`:

```
Command: (setq e1 (entnext))
<Entity name: 2c90520>
```

Delete the entity named by e1:

```
Command: (entdel e1)
<Entity name: 2c90520>
```

Restore the entity named by e1:

```
Command: (entdel e1)
<Entity name: 2c90520>
```

See also:

The handent (page 114) function.

entget

Retrieves an object's (entity's) definition data

```
(entget
  ename [applist]
)
```

Arguments

ename Name of the entity being queried. The *ename* can refer to either a graphical or a nongraphical entity.

applist A list of registered application names.

Return Values

An association list containing the entity definition of *ename*. If you specify the optional *applist* argument, **entget** also returns the extended data associated with the specified applications. Objects in the list are assigned AutoCAD DXF™ group codes for each part of the entity data.

Note that the DXF group codes used by AutoLISP differ slightly from the group codes in a DXF file. The AutoLISP DXF group codes are documented in the *DXF Reference*.

Examples

Assume that the last object created in the drawing is a line drawn from point (1,2) to point (6,5). The following example shows code that retrieves the entity name of the last object with the **entlast** function, and passes that name to **entget**:

```
Command: (entget (entlast))
```

```
((-1 . <Entity name: 1bbd1d0>) (0 . "LINE") (330 . <Entity
name: 1bbd0c8>) (5 . "6A") (100 . "AcDbEntity") (67 . 0) (410
. "Model") (8 . "0") (100 . "AcDbLine") (10 1.0 2.0 0.0) (11
6.0 5.0 0.0) (210 0.0 0.0 1.0))
```

See also:

The entdel (page 67), entlast (page 69), entmod (page 73), entmake (page 70), entnext (page 75), entupd (page 78), and handent (page 114) functions. The Entity Data Functions in the *AutoLISP Developer's Guide*.

entlast

Returns the name of the last nondeleted main object (entity) in the drawing

```
(entlast)
```

The **entlast** function is frequently used to obtain the name of a new entity that has just been added with the **command** function. To be selected, the entity need not be on the screen or on a thawed layer.

Return Values

An entity name; otherwise `nil`, if there are no entities in the current drawing.

Examples

Set variable `e1` to the name of the last entity added to the drawing:

```
Command: (setq e1 (entlast))
<Entity name: 2c90538>
```

If your application requires the name of the last nondeleted entity (main entity or subentity), define a function such as the following and call it instead of **entlast**.

```
(defun lastent (/ a b)
  (if (setq a (entlast))
  Gets last main entity

    (while (setq b (entnext a))
  If subentities follow, loops
  until there are no more
```

```
        (setq a b)
    subentities

      )
    )
    a
    Returns last main entity

  )
    or subentity
```

See also:

The entdel (page 67), entget (page 68), entmod (page 73), entnext (page 75), entsel (page 77), and handent (page 114) functions.

entmake

Creates a new entity in the drawing

```
(entmake
  [elist]
)
```

The **entmake** function can define both graphical and nongraphical entities.

Arguments

elist A list of entity definition data in a format similar to that returned by the **entget** function. The *elist* argument must contain all of the information necessary to define the entity. If any required definition data is omitted, **entmake** returns `nil` and the entity is rejected. If you omit optional definition data (such as the layer), **entmake** uses the default value.

The entity type (for example, `CIRCLE` or `LINE`) must be the first or second field of the *elist*. If entity type is the second field, it can be preceded only by the entity name. The **entmake** function ignores the entity name when creating the new entity. If the *elist* contains an entity handle, **entmake** ignores that too.

Return Values

If successful, **entmake** returns the entity's list of definition data. If **entmake** is unable to create the entity, it returns `nil`.

Completion of a block definition (**entmake** of an endblk) returns the block's name rather than the entity data list normally returned.

Examples

The following code creates a red circle (color 62), centered at (4,4) with a radius of 1. The optional layer and linetype fields have been omitted and therefore assume default values.

```
Command: (entmake '((0 . "CIRCLE") (62 . 1) (10 4.0 4.0 0.0) (40 . 1.0)))
((0 . "CIRCLE") (62 . 1) (10 4.0 4.0 0.0) (40 . 1.0))
```

Notes on Using entmake

You cannot create viewport objects with **entmake**.

A group 66 code is honored only for insert objects (meaning *attributes follow*). For polyline entities, the group 66 code is forced to a value of 1 (meaning *vertices follow*), and for all other entities it takes a default of 0. The only entity that can follow a polyline entity is a vertex entity.

The group code 2 (block name) of a dimension entity is optional for the **entmake** function. If the block name is omitted from the entity definition list, AutoCAD creates a new one. Otherwise, AutoCAD creates the dimension using the name provided.

For legacy reasons, **entmake** ignores DXF group code 100 data for the following entity types:

- AcDbText
- AcDbAttribute
- AcDbAttributeDefinition
- AcDbBlockBegin
- AcDbBlockEnd
- AcDbSequenceEnd
- AcDbBlockReference
- AcDbMInsertBlock
- AcDb2dVertex
- AcDb3dPolylineVertex
- AcDbPolygonMeshVertex
- AcDbPolyFaceMeshVertex

- AcDbFaceRecord
- AcDb2dPolyline
- AcDb3dPolyline
- AcDbArc
- AcDbCircle
- AcDbLine
- AcDbPoint
- AcDbFace
- AcDbPolyFaceMesh
- AcDbPolygonMesh
- AcDbTrace
- AcDbSolid
- AcDbShape
- AcDbViewport

NOTE In AutoCAD 2004 and later releases, the `entmod` function has a new behavior in color operations. DXF group code 62 holds AutoCAD Color Index (ACI) values, but code 420 holds true color values. If the true color value and ACI value conflict, AutoCAD uses the 420 value, so the code 420 value should be removed before attempting to use the code 62 value.

See also:

The entdel (page 67), entget (page 68), entmod (page 73), and handent (page 114) functions. In the *AutoLISP Developer's Guide*, refer to Entity Data Functions for additional information on creating entities in a drawing, Adding an Entity to a Drawing for specifics on using **entmake**, and Creating Complex Entities for information on creating complex entities.

entmakex

Makes a new object or entity, gives it a handle and entity name (but does not assign an owner), and then returns the new entity name

```
(entmakex
  [elist]
```

```
)
```

The **entmakex** function can define both graphical and nongraphical entities.

Arguments

elist A list of entity definition data in a format similar to that returned by the **entget** function. The *elist* argument must contain all of the information necessary to define the entity. If any required definition data is omitted, **entmakex** returns `nil` and the entity is rejected. If you omit optional definition data (such as the layer), **entmakex** uses the default value.

Return Values

If successful, **entmakex** returns the name of the entity created. If **entmakex** is unable to create the entity, the function returns `nil`.

Examples

```
(entmakex '((0 . "CIRCLE") (62 . 1) (10 4.0 3.0 0.0) (40 . 1.0)))
<Entity name: 1d45558>
```

WARNING Objects and entities without owners are not written out to *DWG* or *DXF* files. Be sure to set an owner at some point after using **entmakex**. For example, you can use **dictadd** to set a dictionary to own an object.

See also:

The entmake (page 70) and handent (page 114) functions.

entmod

Modifies the definition data of an object (entity)

```
(entmod
  elist
)
```

The **entmod** function updates database information for the entity name specified by the -1 group in *elist*. The primary mechanism through which AutoLISP updates the database is by retrieving entities with **entget**, modifying the list defining an entity, and updating the entity in the database with

entmod. The **entmod** function can modify both graphical and nongraphical objects.

Arguments

elist A list of entity definition data in a format similar to that returned by the **entget** function.

For entity fields with floating-point values (such as thickness), **entmod** accepts integer values and converts them to floating point. Similarly, if you supply a floating-point value for an integer entity field (such as color number), **entmod** truncates it and converts it to an integer.

Return Values

If successful, **entmod** returns the *elist* supplied to it. If **entmod** is unable to modify the specified entity, the function returns `nil`.

Examples

The following sequence of commands obtains the properties of an entity, and then modifies the entity.

Set the `en1` variable to the name of the first entity in the drawing:

```
Command: (setq en1 (entnext))
<Entity name: 2c90520>
```

Set a variable named `ed` to the entity data of entity `en1`:

```
Command: (setq ed (entget en1))
((-1 . <Entity name: 2c90520>) (0 . "CIRCLE") (5 . "4C") (100
. "AcDbEntity") (67 . 0) (8 . "0") (100 . "AcDbCircle") (10
3.45373 6.21635 0.0) (40 . 2.94827) (210 0.0 0.0 1.0))
```

Changes the layer group in `ed` from layer 0 to layer 1:

```
Command: (setq ed (subst (cons 8 "1") (assoc 8 ed) ed ))
((-1 . <Entity name: 2c90520>) (0 . "CIRCLE") (5 . "4C") (100
. "AcDbEntity") (67 . 0) (8 . "1") (100 . "AcDbCircle") (10
3.45373 6.21635 0.0) (40 . 2.94827) (210 0.0 0.0 1.0))
```

Modify the layer of the `en1` entity in the drawing:

```
Command: (entmod ed)
((-1 . <Entity name: 2c90520>) (0 . "CIRCLE") (5 . "4C") (100
. "AcDbEntity") (67 . 0) (8 . "1") (100 . "AcDbCircle") (10
3.45373 6.21635 0.0) (40 . 2.94827) (210 0.0 0.0 1.0))
```

Restrictions on Using entmod

There are restrictions on the changes the **entmod** function can make:

- An entity's type and handle cannot be changed. If you want to do this, use **entdel** to delete the entity, and then make a new entity with the **command** or **entmake** function.
- The **entmod** function cannot change internal fields, such as the entity name in the -2 group of a seqend entity. Attempts to change such fields are ignored.
- You cannot use the **entmod** function to modify a viewport entity.

You can change an entity's space visibility field to 0 or 1 (except for viewport objects). If you use **entmod** to modify an entity within a block definition, the modification affects all instances of the block in the drawing.

Before performing an **entmod** on vertex entities, you should read or write the polyline entity's header. If the most recently processed polyline entity is different from the one to which the vertex belongs, width information (the 40 and 41 groups) can be lost.

WARNING You can use **entmod** to modify entities within a block definition, but doing so can create a self-referencing block, which will cause AutoCAD to stop.

NOTE In AutoCAD 2004 and later releases, the `entmod` function has a new behavior in color operations. DXF group code 62 holds AutoCAD Color Index (ACI) values, but code 420 holds true color values. If the true color value and ACI value conflict, AutoCAD uses the 420 value, so the code 420 value should be removed before attempting to use the code 62 value. For more information, perform a full installation of AutoCAD and see the *color-util.lsp* file located in the *\Sample\VisualLISP* folder.

See also:

The entdel (page 67), entget (page 68), entmake (page 70), entnext (page 75), and handent (page 114) functions. In the *AutoLISP Developer's Guide*, refer to Modifying an Entity and Entity Data Functions and the Graphics Screen.

entnext

Returns the name of the next object (entity) in the drawing

```
(entnext
  [ename]
)
```

Arguments

ename The name of an existing entity.

Return Values

If **entnext** is called with no arguments, it returns the entity name of the first nondeleted entity in the database. If an *ename* argument is supplied to **entnext,** the function returns the entity name of the first nondeleted entity following *ename* in the database. If there is no next entity in the database, it returns `nil`. The **entnext** function returns both main entities and subentities.

Examples

```
(setq e1 (entnext))     ;
  Sets
e1
   to the name of the first entity in
   the  drawing

(setq e2 (entnext e1)) ;
  Sets
e2
   to the name of the entity
   following
e1
```

Notes

The entities selected by **ssget** are main entities, not attributes of blocks or vertices of polylines. You can access the internal structure of these complex entities by walking through the subentities with **entnext**. Once you obtain a subentity's name, you can operate on it like any other entity. If you obtain the name of a subentity with **entnext**, you can find the parent entity by stepping forward with **entnext** until a seqend entity is found, then extracting the -2 group from that entity, which is the main entity's name.

See also:

The entdel (page 67), entget (page 68), entmake (page 70), entnext (page 75), and handent (page 114) functions.

entsel

Prompts the user to select a single object (entity) by specifying a point

```
(entsel
  [msg]
)
```

Arguments

msg A prompt string to be displayed to users. If omitted, **entsel** prompts with the message, "Select object."

Return Values

A list whose first element is the entity name of the chosen object and whose second element is the coordinates (in terms of the current UCS) of the point used to pick the object.

The pick point returned by **entsel** does not represent a point that lies on the selected object. The point returned is the location of the crosshairs at the time of selection. The relationship between the pick point and the object will vary depending on the size of the pickbox and the current zoom scale.

Examples

The following AutoCAD command sequence illustrates the use of the **entsel** function and the list returned:

```
Command: line
From point: 1,1
To point: 6,6
To point: ENTER
Command: (setq e (entsel "Please choose an object: "))
Please choose an object: 3,3
(<Entity name: 60000014> (3.0 3.0 0.0))
```

When operating on objects, you may want to simultaneously select an object and specify the point by which it was selected. Examples of this in AutoCAD can be found in Object Snap and in the BREAK, TRIM, and EXTEND commands

in the *Command Reference*. The **entsel** function allows AutoLISP programs to perform this operation. It selects a single object, requiring the selection to be a pick point. The current Osnap setting is ignored by this function unless you specifically request it while you are in the function. The **entsel** function honors keywords from a preceding call to **initget**.

See also:

The entget (page 68), entmake (page 70), entnext (page 75), handent (page 114), and initget (page 118) functions.

entupd

Updates the screen image of an object (entity)

```
(entupd
  ename
)
```

Arguments

ename The name of the entity to be updated on the screen.

Return Values

The entity (*ename*) updated; otherwise `nil`, if nothing was updated.

Examples

Assuming that the first entity in the drawing is a 3D polyline with several vertices, the following code modifies and redisplays the polyline:

```
(setq e1 (entnext))       ;
  Sets
 e1
  to the polyline's entity name

(setq e2 (entnext e1))    ;
  Sets
 e2
  to its first vertex
```

```
(setq ed (entget e2))     ;
  Sets
 ed
  to the vertex data

(setq ed
  (subst '(10 1.0 2.0)
    (assoc 10 ed)         ;
  Changes the vertex's location in
 ed
    ed                    ;
  to point (
1,2
  )

  )
)
(entmod ed)               ;
  Moves the vertex in the drawing

(entupd e1)               ;
  Regenerates the polyline entity
 e1
```

Updating Polylines and Blocks

When a 3D (or old-style) polyline vertex or block attribute is modified with **entmod**, the entire complex entity is not updated on the screen. The **entupd** function can be used to cause a modified polyline or block to be updated on the screen. This function can be called with the entity name of any part of the polyline or block; it need not be the head entity. While **entupd** is intended for polylines and blocks with attributes, it can be called for any entity. It always regenerates the entity on the screen, including all subentities.

NOTE If **entupd** is used on a nested entity (an entity within a block) or on a block that contains nested entities, some of the entities might not be regenerated. To ensure complete regeneration, you must invoke the REGEN command in the *Command Reference*.

See also:

The entget (page 68), entmod (page 73), entnext (page 75), and handent (page 114) functions.

eq

Determines whether two expressions are identical

```
(eq
  expr1 expr2
)
```

The **eq** function determines whether *expr1* and *expr2* are bound to the same object (by **setq**, for example).

Arguments

expr1 The expression to be compared.

expr2 The expression to compare with *expr1*.

Return Values

`T` if the two expressions are identical; otherwise `nil`.

Examples

Given the following assignments:

```
(setq f1 '(a b c))
(setq f2 '(a b c))
(setq f3 f2)
```

Compare `f1` and `f3`:
`Command:` **(eq f1 f3)**
`nil`

eq returns `nil` because `f1` and `f3`, while containing the same value, do not refer to the same list.

Compare `f3` and `f2`:
`Command:` **(eq f3 f2)**
`T`

eq returns `T` because `f3` and `f2` refer to the same list.

See also:

The = (equal to) (page 5) and equal (page 81) functions.

equal

Determines whether two expressions are equal

```
(equal
  expr1 expr2 [fuzz]
)
```

Arguments

expr1 The expression to be compared.

expr2 The expression to compare with *expr1*.

fuzz A real number defining the maximum amount by which *expr1* and *expr2* can differ and still be considered equal.

When comparing two real numbers (or two lists of real numbers, as in points), the two identical numbers can differ slightly if different methods are used to calculate them. You can specify a *fuzz* amount to compensate for the difference that may result from the different methods of calculation.

Return Values

`T` if the two expressions are equal (evaluate to the same value); otherwise `nil`.

Examples

Given the following assignments:

```
(setq f1 '(a b c))
(setq f2 '(a b c))
(setq f3 f2)
(setq a 1.123456)
(setq b 1.123457)
```

Compare `f1` to `f3`:
`Command:` **(equal f1 f3)**
`T`

Compare `f3` to `f2`:
`Command:` **(equal f3 f2)**

```
T
```

Compare `a` to `b`:

```
Command: (equal a b)
nil
```

The `a` and `b` variables differ by .000001.

Compare `a` to `b`:, with *fuzz* argument of .000001:

```
Command: (equal a b 0.000001)
T
```

The `a` and `b` variables differ by an amount equal to the specified *fuzz* factor, so **equal** considers the variables equal.

Comparing the eq and equal Functions

If the **eq** function finds that two lists or atoms are the same, the **equal** function also finds them to be the same.

Any *atoms* that the **equal** function determines to be the same are also found equivalent by **eq**. However, two *lists* that **equal** determines to be the same may be found to be different according to the **eq** function.

See also:

The= (equal to) (page 5)and eq (page 80) functions.

error

A user-definable error-handling function

```
(*error*
  string
)
```

If ***error*** is not `nil`, it is executed as a function whenever an AutoLISP error condition exists. AutoCAD passes one argument to ***error***, which is a string containing a description of the error.

Your ***error*** function can include calls to the **command** function without arguments (for example, `(command)`). This will cancel a previous AutoCAD command called with the **command** function.

Return Values

This function does not return, except when using <Undefined Cross-Reference> (page 226).

Examples

The following function does the same thing that the AutoLISP standard error handler does. It prints the word "error," followed by a description:

```
(defun *error* (msg)
  (princ "error: ")
  (princ msg)
  (princ)
)
```

See also:

The vl-exit-with-error (page 225), vl-exit-with-value (page 226), vl-catch-all-apply (page 216), vl-catch-all-error-message (page 217), and vl-catch-all-error-p (page 218) functions.

eval

Returns the result of evaluating an AutoLISP expression

```
(eval
  expr
)
```

Arguments

expr The expression to be evaluated.

Return Values

The result of the expression, after evaluation.

Examples

First, set some variables:

```
Command: (setq a 123)
123
Command: (setq b 'a)
A
```

Now evaluate some expressions:

```
Command: (eval 4.0)
4.0
Command: (eval (abs -10))
10
Command: (eval a)
123
Command: (eval b)
123
```

exit

Forces the current application to quit

```
(exit)
```

If **exit** is called, it returns the error message quit/exit abort and returns to the AutoCAD Command prompt.

See also:

The quit (page 163) function.

exp

Returns the constant *e* (a real number) raised to a specified power (the natural antilog)

```
(exp
  num
)
```

Arguments

num A real number.

Return Values

A real (*num*), raised to its natural antilogarithm.

Examples

```
Command: (exp 1.0)
2.71828
```

```
Command: (exp 2.2)
9.02501
Command: (exp -0.4)
0.67032
```

expand

Allocates additional memory for AutoLISP

```
(expand
   n-expand
)
```

Arguments

n-expand An integer indicating the amount of additional memory to be allocated. Memory is allocated as follows:

- *n-alloc* free symbols
- *n-alloc* free strings
- *n-alloc* free usubrs
- *n-alloc* free reals
- *n-alloc* * *n-expand* cons cells

where *n-alloc* is the current segment size.

Return Values

An integer indicating the number of free conses divided by *n-alloc*.

Examples

Set the segment size to 100:

```
(alloc 100)
1000
```

Allocate memory for two additional segments:

```
(expand 2)
82
```

This ensures that AutoLISP now has memory available for at least 200 additional symbols, strings, usubrs and reals each, and 8200 free conses.

See also:

The alloc (page 21) function.

expt

Returns a number raised to a specified power

```
(expt
  number power
)
```

Arguments

number Any number.

power The power to raise *number* to.

Return Values

If both arguments are integers, the result is an integer; otherwise, the result is a real.

Examples

```
Command: (expt 2 4)
16
Command: (expt 3.0 2.0)
9.0
```

F Functions

findfile

Searches the AutoCAD library path for the specified file or directory

```
(findfile
  filename
)
```

The **findfile** function makes no assumption about the file type or extension of *filename*. If *filename* does not specify a drive/directory prefix, **findfile** searches the AutoCAD library path. If a drive/directory prefix is supplied, **findfile** looks only in that directory.

Arguments

filename Name of the file or directory to be searched for.

Return Values

A string containing the fully qualified file name; otherwise `nil`, if the specified file or directory is not found.

The file name returned by **findfile** is suitable for use with the **open** function.

Examples

If the current directory is */ MyUtilities/lsp* and it contains the file *abc.lsp*, the following function call retrieves the path name:

```
Command: (findfile "abc.lsp")
"/MyUtilities/Lsp/abc.lsp"
```

If you are editing a drawing in the */ My Utilities/Support* directory, and the ACAD system variable is set to */ My Utilities/Support*, and the file *xyz.txt* exists only in the */ My Utilities/Support* directory, then the following command retrieves the path name:

```
Command: (findfile "xyz.txt")
"/MyUtilities/Support/xyz.txt"
```

If the file *nosuch* is not present in any of the directories on the library search path, **findfile** returns `nil`:

```
Command: (findfile "nosuch")
nil
```

fix

Returns the conversion of a real number into the nearest smaller integer

```
(fix
  number
)
```

The **fix** function truncates *number* to the nearest integer by discarding the fractional portion.

Arguments

number A real number.

Return Values

The integer derived from *number*.

If *number* is larger than the largest possible integer (+2,147,483,647 or -2,147,483,648 on a 32-bit platform), **fix** returns a truncated real (although integers transferred between AutoLISP and AutoCAD are restricted to 16-bit values).

Examples

```
Command: (fix 3)
3
Command: (fix 3.7)
3
```

float

Returns the conversion of a number into a real number

```
(float
  number
)
```

Arguments

number Any number.

Return Values

The real number derived from *number*.

Examples

```
Command: (float 3)
3.0
Command: (float 3.75)
3.75
```

foreach

Evaluates expressions for all members of a list

```
(foreach
  name list [expr
...
  ]
)
```

The **foreach** function steps through a list, assigning each element in the list to a variable, and evaluates each expression for every element in the list. Any number of expressions can be specified.

Arguments

name Variable that each element in the list will be assigned to.

list List to be stepped through and evaluated.

expr Expression to be evaluated for each element in *list*.

Return Values

The result of the last *expr* evaluated. If no *expr* is specified, **foreach** returns `nil`.

Examples

Print each element in a list:

```
Command: (foreach n '(a b c) (print n))
A
B
C C
```

foreach prints each element in the list and returns `C`, the last element. This command is equivalent to the following sequence of commands, except that **foreach** returns the result of only the last expression evaluated:

```
(print a)
(print b)
(print c)
```

function

Tells the AutoLISP compiler to link and optimize an argument as if it were a built-in function

```
(function
  symbol | lambda-expr
)
```

The **function** function is identical to the **quote** function, except it tells the AutoLISP compiler to link and optimize the argument as if it were a built-in function or **defun**.

Arguments

symbol A symbol naming a function.

lambda-expr An expression of the following form:

`(LAMBDA` *arguments {S-expression}**`)`

Return Values

The result of the evaluated expression.

Examples

The AutoLISP compiler cannot optimize the quoted **lambda** expression in the following code:

```
(mapcar
  '(lambda (x) (* x x))
       '(1 2 3))
```

After adding the **function** function to the expression, the compiler can optimize the **lambda** expression. For example:

```
(mapcar
   (function (lambda (x) (* x x)))
      '(1 2 3))
```

G Functions

gc

Forces a garbage collection, which frees up unused memory

```
(gc)
```

See also:

The Memory Management Functions topic in the *AutoLISP Developer's Guide*.

gcd

Returns the greatest common denominator of two integers

```
(gcd
  int1 int2
)
```

Arguments

int1 An integer; must be greater than 0.

int2 An integer; must be greater than 0.

Return Values

An integer representing the greatest common denominator between *int1* and *int2*.

Examples

```
Command: (gcd 81 57)
3
Command: (gcd 12 20)
4
```

getangle

Pauses for user input of an angle, and returns that angle in radians

```
(getangle
  [pt] [msg]
)
```

Arguments

pt A 2D base point in the current UCS.

The *pt* argument, if specified, is assumed to be the first of two points, so the user can show AutoLISP the angle by pointing to one other point. You can supply a 3D base point, but the angle is always measured in the current construction plane.

msg A string to be displayed to prompt the user.

Return Values

The angle specified by the user, in radians.

The **getangle** function measures angles with the zero-radian direction (set by the ANGBASE system variable in the *Command Reference*) with angles increasing in the counterclockwise direction. The returned angle is expressed in radians with respect to the current construction plane (the *XY* plane of the current UCS, at the current elevation).

Examples

The following code examples show how different arguments can be used with **getangle**:

```
Command: (setq ang (getangle))
Command: (setq ang (getangle '(1.0 3.5)))
Command: (setq ang (getangle "Which way? "))
Command: (setq ang (getangle '(1.0 3.5) "Which way? "))
```

Usage Notes

Users can specify an angle by entering a number in the AutoCAD current angle units format. Although the current angle units format might be in degrees, grads, or some other unit, this function always returns the angle in radians. The user can also show AutoLISP the angle by pointing to two 2D locations in the drawing area. AutoCAD draws a rubber-band line from the first point to the current crosshairs position to help you visualize the angle.

It is important to understand the difference between the input angle and the angle returned by **getangle**. Angles that are passed to **getangle** are based on the current settings of ANGDIR and ANGBASE in the *Command Reference*. However, once an angle is provided, it is measured in a counterclockwise direction (ignoring ANGDIR) with zero radians as the current setting of ANGBASE.

The user cannot enter another AutoLISP expression as the response to a **getangle** request.

See also:

The illustration and comparison to the getorient (page 101) function, the initget (page 118) function, and The getxxx Functions in the *AutoLISP Developer's Guide*.

getcfg

Retrieves application data from the AppData section of the *acad.cfg* file

```
(getcfg
  cfgname
)
```

Arguments

cfgname A string (maximum length of 496 characters) naming the section and parameter value to retrieve.

The *cfgname* argument must be a string of the following form:

```
"AppData/
  application_name
/
  section_name
/.../
  param_name
"
```

Return Values

Application data, if successful. If *cfgname* is not valid, **getcfg** returns `nil`.

Examples

Assuming the WallThk parameter in the AppData/ArchStuff section has a value of 8, the following command retrieves that value:

```
Command: (getcfg "AppData/ArchStuff/WallThk")
"8"
```

See also:

The setcfg (page 173) function.

getcname

Retrieves the localized or English name of an AutoCAD command

```
(getcname
  cname
)
```

Arguments

cname The localized or underscored English command name; must be 64 characters or less in length.

Return Values

If *cname* is not preceded by an underscore (assumed to be the localized command name), **getcname** returns the underscored English command name. If *cname* is preceded by an underscore, **getcname** returns the localized command name. This function returns `nil` if *cname* is not a valid command name.

Examples

In a French version of AutoCAD, the following is true.

```
(getcname "ETIRER")
returns
"_STRETCH"
(getcname "_STRETCH")
returns
"ETIRER"
```

getcorner

Pauses for user input of a rectangle's second corner

```
(getcorner
  pt [msg]
)
```

The **getcorner** function takes a base point argument, based on the current UCS, and draws a rectangle from that point as the user moves the crosshairs on the screen.

The user cannot enter another AutoLISP expression in response to a **getcorner** request.

Arguments

pt A point to be used as the base point.

msg A string to be displayed to prompt the user.

Return Values

The **getcorner** function returns a point in the current UCS, similar to **getpoint**. If the user supplies a 3D point, its *Z* coordinate is ignored. The current elevation is used as the *Z* coordinate.

Examples

```
Command: (getcorner '(7.64935 6.02964 0.0))
(17.2066 1.47628 0.0)
Command: (getcorner '(7.64935 6.02964 0.0) "Pick a corner")
Pick a corner(15.9584 2.40119 0.0)
```

See also:

The initget (page 118) function. The getxxx Functions in the *AutoLISP Developer's Guide*.

getdist

Pauses for user input of a distance

```
(getdist
  [pt] [msg]
)
```

The user can specify the distance by selecting two points, or by specifying just the second point, if a base point is provided. The user can also specify a distance by entering a number in the AutoCAD current distance units format. Although the current distance units format might be in feet and inches (architectural), the **getdist** function always returns the distance as a real.

The **getdist** function draws a rubber-band line from the first point to the current crosshairs position to help the user visualize the distance.

The user cannot enter another AutoLISP expression in response to a **getdist** request.

Arguments

pt A 2D or 3D point to be used as the base point in the current UCS. If *pt* is provided, the user is prompted for the second point.

msg A string to be displayed to prompt the user. If no string is supplied, AutoCAD does not display a message.

Return Values

A real number. If a 3D point is provided, the returned value is a 3D distance. However, setting the 64 bit of the **initget** function instructs **getdist** to ignore the *Z* component of 3D points and to return a 2D distance.

Examples

```
(setq dist (getdist))
(setq dist (getdist '(1.0 3.5)))
(setq dist (getdist "How far "))
(setq dist (getdist '(1.0 3.5) "How far? "))
```

See also:

The initget (page 118) function. The getxxx Functions in the *AutoLISP Developer's Guide*.

getenv

Returns the string value assigned to a system environment variable

```
(getenv
  variable-name
)
```

Arguments

variable-name A string specifying the name of the variable to be read. Environment variable names must be spelled and cased exactly as they are stored in the system registry.

Return Values

A string representing the value assigned to the specified system variable. If the variable does not exist, **getenv** returns `nil`.

Examples

Assume the system environment variable `ACAD` is set to */acad/support* and there is no variable named `NOSUCH`.

```
Command: (getenv "ACAD")
"/acad/support"
Command: (getenv "NOSUCH")
nil
```

Assume that the `MaxArray` environment variable is set to 10000:

```
Command: (getenv "MaxArray")
"10000"
```

See also:

The setenv (page 174)function.

getfiled

Prompts the user for a file name with the standard AutoCAD file dialog box, and returns that file name

```
(getfiled
  title default ext flags
)
```

The **getfiled** function displays a dialog box containing a list of available files of a specified extension type. You can use this dialog box to browse through different drives and directories, select an existing file, or specify the name of a new file.

Arguments

title A string specifying the dialog box label.

default A default file name to use; can be a null string (`""`).

ext The default file name extension. If *ext* is passed as a null string (`""`), it defaults to `*` (all file types).

If the file type `dwg` is included in the *ext* argument, the **getfiled** function displays an image preview in the dialog box.

flags An integer value (a bit-coded field) that controls the behavior of the dialog box. To set more than one condition at a time, add the values together to create a *flags* value between 0 and 15. The following *flags* arguments are recognized by **getfiled**:

1 (bit 0) Prompt for the name of a new file to create. Do not set this bit when you prompt for the name of an existing file to open. In the latter case, if the user enters the name of a file that doesn't exist, the dialog box displays an error message at the bottom of the box.

If this bit is set and the user chooses a file that already exists, AutoCAD displays an alert box and offers the choice of proceeding with or canceling the operation.

4 (bit 2) Let the user enter an arbitrary file name extension, or no extension at all.

If this bit is not set, **getfiled** accepts only the extension specified in the *ext* argument and appends this extension to the file name if the user doesn't enter it in the File text box.

8 (bit 3) If this bit is set and bit 0 is not set, **getfiled** performs a library search for the file name entered. If it finds the file and its directory in the library search path, it strips the path and returns only the file name. (It does not strip the path name if it finds that a file of the same name is in a different directory.)

If this bit is not set, **getfiled** returns the entire file name, including the path name.

Set this bit if you use the dialog box to open an existing file whose name you want to save in the drawing (or other database).

16 (bit 4) If this bit is set, or if the *default* argument ends with a path delimiter, the argument is interpreted as a path name only. The **getfiled** function assumes that there is no default file name. It displays the path in the Look in: line and leaves the File name box blank.

32 (bit 5) If this bit is set and bit 0 is set (indicating that a new file is being specified), users will not be warned if they are about to overwrite an existing file. The alert box to warn users that a file of the same name already exists will not be displayed; the old file will just be replaced.

64 (bit 6) Do not transfer the remote file if the user specifies a URL.

128 (bit 7) Do not allow URLs at all.

Return Values

If the dialog box obtains a file name from the user, **getfiled** returns a string that specifies the file name; otherwise, it returns `nil`.

Examples

The following call to **getfiled** displays the Select a Lisp File dialog box:

```
(getfiled "Select a Lisp File"
"/Applications/Autodesk/AutoCAD 2013/AutoCAD 2013.app/"
"lsp" 8)
```

getint

Pauses for user input of an integer, and returns that integer

```
(getint
  [msg]
)
```

Values passed to **getint** can range from -32,768 to +32,767. If the user enters something other than an integer, **getint** displays the message, "Requires an integer value," and allows the user to try again. The user cannot enter another AutoLISP expression as the response to a **getint** request.

Arguments

msg A string to be displayed to prompt the user; if omitted, no message is displayed.

Return Values

The integer specified by the user; otherwise `nil`, if the user presses ENTER without entering an integer.

Examples

```
Command: (setq num (getint))
15
15
Command: (setq num (getint "Enter a number:"))
Enter a number: 25
25
Command: (setq num (getint))
15.0
```

```
Requires an integer value.
15
15
```

See also:

The initget (page 118) function. The getxxx Functions in the *AutoLISP Developer's Guide*.

getkword

Pauses for user input of a keyword, and returns that keyword

```
(getkword
  [msg]
)
```

Valid keywords are set prior to the **getkword** call with the **initget** function. The user cannot enter another AutoLISP expression as the response to a **getkword** request.

Arguments

msg A string to be displayed to prompt the user; if omitted, **getkword** does not display a prompting message.

Return Values

A string representing the keyword entered by the user; otherwise `nil`, if the user presses ENTER without typing a keyword. The function also returns `nil` if it was not preceded by a call to **initget** to establish one or more keywords.

If the user enters a value that is not a valid keyword, **getkword** displays a warning message and prompts the user to try again.

Examples

The following example shows an initial call to **initget** that sets up a list of keywords (Yes and No) and disallows null input (*bits* value equal to 1) to the **getkword** call that follows:

```
Command: (initget 1 "Yes No")
nil
Command: (setq x (getkword "Are you sure? (Yes or No) "))
Are you sure? (Yes or No) yes
```

```
"Yes"
```

The following sequence illustrates what happens if the user enters invalid data in response to **getkword**:

```
Command: (initget 1 "Yes No")
nil
Command: (setq x (getkword "Are you sure? (Yes or No) "))
Are you sure? (Yes or No) Maybe
Invalid option keyword.
Are you sure? (Yes or No) yes
"Yes"
```

The user's response was not one of the keywords defined by the preceding **initget**, so **getkword** issued an error message and then prompted the user again with the string supplied in the *msg* argument.

See also:

The initget (page 118) function. The getxxx Functions in the *AutoLISP Developer's Guide*.

getorient

Pauses for user input of an angle, and returns that angle in radians

```
(getorient
  [pt] [msg]
)
```

The **getorient** function measures angles with the zero-radian direction to the right (east) and angles that are increasing in the counterclockwise direction. The angle input by the user is based on the current settings of ANGDIR and ANGBASE, but once an angle is provided, it is measured in a counterclockwise direction, with zero radians being to the right (ignoring ANGDIR and ANGBASE). Therefore, some conversion must take place if you select a different zero-degree base or a different direction for increasing angles by using the UNITS command or the ANGBASE and ANGDIR system variables in the *Command Reference*.

Use **getangle** when you need a rotation amount (a relative angle). Use **getorient** to obtain an orientation (an absolute angle).

The user cannot enter another AutoLISP expression as the response to a **getorient** request.

Arguments

pt A 2D base point in the current UCS.

The *pt* argument, if specified, is assumed to be the first of two points, so that the user can show AutoLISP the angle by pointing to one other point. You can supply a 3D base point, but the angle is always measured in the current construction plane.

msg A string to be displayed to prompt the user.

Return Values

The angle specified by the user, in radians, with respect to the current construction plane.

Examples

```
Command: (setq pt1 (getpoint "Pick point: "))
(4.55028 5.84722 0.0)
Command: (getorient pt1 "Pick point: ")
5.61582
```

getpoint

Pauses for user input of a point, and returns that point

```
(getpoint
  [pt] [msg]
)
```

The user can specify a point by pointing or by entering a coordinate in the current units format. If the *pt* argument is present, AutoCAD draws a rubber-band line from that point to the current crosshairs position.

The user cannot enter another AutoLISP expression in response to a **getpoint** request.

Arguments

pt A 2D or 3D base point in the current UCS.

Note that **getpoint** will accept a single integer or real number as the *pt* argument, and use the AutoCAD direct distance entry mechanism to determine a point. This mechanism uses the value of the LASTPOINT system variable in

the *Command Reference* as the starting point, the *pt* input as the distance, and the current cursor location as the direction from LASTPOINT. The result is a point that is the specified number of units away from LASTPOINT in the direction of the current cursor location.

msg A string to be displayed to prompt the user.

Return Values

A 3D point, expressed in terms of the current UCS.

Examples

```
(setq p (getpoint))
(setq p (getpoint "Where? "))
(setq p (getpoint '(1.5 2.0) "Second point: "))
```

See also:

The getcorner (page 94) and initget (page 118) functions. The getxxx Functions in the *AutoLISP Developer's Guide*.

getpropertyvalue

Returns the current value of an entity's property.

```
(getpropertyvalue
  ename propertyname [or collectionName index name]
)
```

Arguments

ename Name of the entity being queried. The ename can refer to either a graphical or a non-graphical entity.

propertyname Name of the property being queried. For a list of all the valid property names of a given object, use `dumpallproperties`.

collectionName If the object is a collection object, the Collection name is passed here.

index The collection index being queried.

name The name of the property within the collection being queried.

Return Values

The value of the entity's property.

Examples

The following example demonstrates how to get the current radius value of a circle.

```
Command: (command "_circle" "2,2" 2)
nil

Command: (getpropertyvalue (entlast) "radius")
2.0
```

See also:

DumpAllProperties (page 64)
IsPropertyReadOnly (page 123)
SetPropertyValue (page 175)

getreal

Pauses for user input of a real number, and returns that real number

```
(getreal
  [msg]
)
```

The user cannot enter another AutoLISP expression as the response to a **getreal** request.

Arguments

msg A string to be displayed to prompt the user.

Return Values

The real number entered by the user.

Examples

```
(setq val (getreal))
(setq val (getreal "Scale factor: "))
```

See also:

The initget (page 118) function. The getxxx Functions in the *AutoLISP Developer's Guide*.

getstring

Pauses for user input of a string, and returns that string

```
(getstring
  [cr]
  [msg]
)
```

The user cannot enter another AutoLISP expression as the response to a **getstring** request.

Arguments

cr If supplied and not `nil`, this argument indicates that users can include blanks in their input string (and must terminate the string by pressing Enter). Otherwise, the input string is terminated by entering a space or pressing Enter.

msg A string to be displayed to prompt the user.

Return Values

The string entered by the user; otherwise `nil`, if the user pressed Enter without typing a string.

If the string is longer than 132 characters, **getstring** returns only the first 132 characters of the string. If the input string contains the backslash character (\), **getstring** converts it to two backslash characters (\\). This allows you to use returned values containing file name paths in other functions.

Examples

```
Command: (setq s (getstring "What's your first name? "))
What's your first name? Gary
"Gary"
Command: (setq s (getstring T "What's your full name? "))
What's your full name? Gary Indiana Jones
"Gary Indiana Jones"
Command: (setq s (getstring T "Enter filename: "))
Enter filename: /myutilities/support/xyz.txt
```

See also:

The initget (page 118) function. The getxxx Functions in the *AutoLISP Developer's Guide*.

getvar

Retrieves the value of an AutoCAD system variable

```
(getvar
  varname
)
```

Arguments

varname A string or symbol that names a system variable. See the *Command Reference* for a list of current AutoCAD system variables.

Return Values

The value of the system variable; otherwise `nil`, if *varname* is not a valid system variable.

Examples

Get the current value of the fillet radius:

```
Command: (getvar 'FILLETRAD)
0.25
```

See also:

The setvar (page 177) function.

graphscr

Displays the AutoCAD graphics screen

NOTE This function is supported on Mac OS, but does not affect AutoCAD.

```
(graphscr)
```

This function is equivalent to the GRAPHSCR command in the *Command Reference* or pressing the Flip Screen function key. The **textscr** function is the complement of **graphscr**.

Returns

```
nil
```

See also:

The textscr (page 206) function.

grclear

Clears the current viewport (obsolete function)

```
(grclear)
```

Returns

```
nil
```

grdraw

Draws a vector between two points, in the current viewport

```
(grdraw
  from to color [highlight]
)
```

Arguments

from 2D or 3D points (lists of two or three reals) specifying one endpoint of the vector in terms of the current UCS. AutoCAD clips the vector to fit the screen.

to 2D or 3D points (lists of two or three reals) specifying the other endpoint of the vector in terms of the current UCS. AutoCAD clips the vector to fit the screen.

color An integer identifying the color used to draw the vector. A -1 signifies *XOR ink*, which complements anything it draws over and which erases itself when overdrawn.

highlight An integer, other than zero, indicating that the vector is to be drawn using the default highlighting method of the display device (usually dashed).

If *highlight* is omitted or is zero, **grdraw** uses the normal display mode.

Return Values

`nil`

See also:

The grvecs (page 112) function for a routine that draws multiple vectors.

grread

Reads values from any of the AutoCAD input devices

```
(grread
  [track] [allkeys [curtype]]
)
```

Only specialized AutoLISP routines need this function. Most input to AutoLISP should be obtained through the various **get*xxx*** functions.

Arguments

track If supplied and not `nil`, this argument enables the return of coordinates from a pointing device as it is moved.

allkeys An integer representing a code that tells **grread** what functions to perform. The *allkeys* bit code values can be added together for combined functionality. The following values can be specified:

1 (bit 0) Return *drag mode* coordinates. If this bit is set and the user moves the pointing device instead of selecting a button or pressing a key, **grread** returns a list where the first member is a type 5 and the second member is the *(X,Y)* coordinates of the current pointing device (mouse or digitizer) location. This is how AutoCAD implements dragging.

2 (bit 1) Return all key values, including function and cursor key codes, and don't move the cursor when the user presses a cursor key.

4 (bit 2) Use the value passed in the *curtype* argument to control the cursor display.

8 (bit 3) Don't display the error: console break message when the user presses Esc.

curtype An integer indicating the type of cursor to be displayed. The *allkeys* value for bit 2 must be set for the *curtype* values to take effect. The *curtype* argument affects only the cursor type during the current **grread** function call. You can specify one of the following values for *curtype*:

0 Display the normal crosshairs.

1 Do not display a cursor (no crosshairs).

2 Display the object-selection "target" cursor.

Return Values

The **grread** function returns a list whose first element is a code specifying the type of input. The second element of the list is either an integer or a point, depending on the type of input. The return values are listed in the following table:

grread return values

First element		**Second element**	
Value	**Type of input**	**Value**	**Description**
2	Keyboard input	varies	Character code
3	Selected point	3D point	Point coordinates
5	Pointing device (returned only if tracking is enabled)	3D point	Drag mode coordinate
6	BUTTONS menu item	0 to 999 1000 to 1999 2000 to 2999 3000 to 3999	BUTTONS1 menu button no. BUTTONS2 menu button no. BUTTONS3 menu button no. BUTTONS4 menu button no.

grread return values			
First element		**Second element**	
Value	**Type of input**	**Value**	**Description**
11	AUX menu item	0 to 999 1000 to 1999 2000 to 2999 3000 to 3999	AUX1 menu button no. AUX2 menu button no. AUX3 menu button no. AUX4 menu button no.
12	Pointer button (follows a type 6 or type 11 return)	3D point	Point coordinates

Handling User Input with grread

Entering Esc while a **grread** is active aborts the AutoLISP program with a keyboard break (unless the *allkeys* argument has disallowed this). Any other input is passed directly to **grread**, giving the application complete control over the input devices.

If the user presses the pointer button within a screen menu or pull-down menu box, **grread** returns a type 6 or type 11 code, but in a subsequent call, it does not return a type 12 code: the type 12 code follows type 6 or type 11 only when the pointer button is pressed while it is in the drawing area.

It is important to clear the code 12 data from the buffer before attempting another operation with a pointer button or an auxiliary button. To accomplish this, perform a nested **grread** like this:

```
(setq code_12 (grread (setq code (grread))))
```

This sequence captures the value of the code 12 list as streaming input from the device.

grtext

Writes text to the status line or to screen menu areas

NOTE This function is supported on Mac OS, but does not affect AutoCAD.

```
(grtext
  [box text [highlight]]
)
```

This function displays the supplied text in the menu area; it does not change the underlying menu item. The **grtext** function can be called with no arguments to restore all text areas to their standard values.

Arguments

box An integer specifying the location in which to write the text.

text A string that specifies the text to be written to the screen menu or status line location. The *text* argument is truncated if it is too long to fit in the available area.

highlight An integer that selects or deselects a screen menu location.

If called without arguments, **grtext** restores all text areas to their standard values. If called with only one argument, **grtext** results in an error.

Return Values

The string passed in the *text* argument, if successful, and `nil` if unsuccessful or no arguments are supplied.

Screen Menu Area

Setting *box* to a positive or zero value specifies a screen menu location. Valid *box* values range from 0 to the highest-numbered screen menu box minus 1. The SCREENBOXES system variable in the *Command Reference* reports the maximum number of screen menu boxes. If the *highlight* argument is supplied as a positive integer, **grtext** highlights the text in the designated box. Highlighting a box automatically dehighlights any other box already highlighted. If *highlight* is zero, the menu item is dehighlighted. If *highlight* is a negative number, it is ignored. On some platforms, the text must first be written without the *highlight* argument and then must be highlighted. Highlighting of a screen menu location works only when the cursor is not in that area.

Status Line Area

If **grtext** is called with a *box* value of -1, it writes the text into the mode status line area. The length of the mode status line differs from display to display (most allow at least 40 characters). The following code uses the **$(linelen)** DIESEL expression to report the length of the mode status area.

```
(setq modelen (menucmd "M=$(linelen)"))
```

If a *box* value of -2 is used, **grtext** writes the text into the coordinate status line area. If coordinate tracking is turned on, values written into this field are overwritten as soon as the pointer sends another set of coordinates. For both -1 and -2 *box* values, the *highlight* argument is ignored.

grvecs

Draws multiple vectors in the drawing area

```
(grvecs
  vlist [trans]
)
```

Arguments

vlist A vector list is comprosed of a series of optional color integers and two point lists. See below for details on how to format *vlist*.

trans A transformation matrix used to change the location or proportion of the vectors defined in your vector list. This matrix is a list of four lists of four real numbers.

Return Values

```
nil
```

Vector List Format

The format for *vlist* is as follows:

```
([color1] from1 to1 [color2] from2 to2 ...)
```

The color value applies to all succeeding vectors until *vlist* specifies another color. AutoCAD colors are in the range 0-255. If the color value is greater than 255, succeeding vectors are drawn in *XOR ink*, which complements anything it draws over and which erases itself when overdrawn. If the color value is less than zero, the vector is highlighted. Highlighting depends on the display device. Most display devices indicate highlighting by a dashed line, but some indicate it by using a distinctive color.

A pair of point lists, *from* and *to*, specify the endpoints of the vectors, expressed in the current UCS. These can be 2D or 3D points. You must pass these points as pairs—two successive point lists—or the **grvecs** call will fail.

AutoCAD clips the vectors as required to fit on the screen.

Examples

The following code draws five vertical lines in the drawing area, each a different color:

```
(grvecs '(1 (1 2)(1 5)
  Draws a red line from (
1,2
  ) to (
1,5
  )

          2 (2 2)(2 5)
  Draws a yellow line from (
2,2
  ) to (
2,5
  )

          3 (3 2)(3 5)
  Draws a green line from (
3,2
  ) to (
3,5
  )

          4 (4 2)(4 5)
  Draws a cyan line from (
4,2
  ) to (
4,5
  )

          5 (5 2)(5 5)
  Draws a blue line from (
```

```
5,2
  ) to (
5,5
  )

) )
```

The following matrix represents a uniform scale of 1.0 and a translation of 5.0,5.0,0.0. If this matrix is applied to the preceding list of vectors, they will be offset by 5.0,5.0,0.0.

```
'((1.0 0.0 0.0 5.0)
   (0.0 1.0 0.0 5.0)
   (0.0 0.0 1.0 0.0)
   (0.0 0.0 0.0 1.0)
)
```

See also:

The nentselp (page 151) function for more information on transformation matrixes and the grdraw (page 107) function for a routine that draws a vector between two points.

H Functions

handent

Returns an object (entity) name based on its handle

```
(handent
  handle
)
```

The **handent** function returns the entity name of both graphic and nongraphic entities.

Arguments

handle A string identifying an entity handle.

Return Values

If successful, **handent** returns the entity name associated with *handle* in the current editing session. If **handent** is passed an invalid handle or a handle not used by any entity in the current drawing, it returns `nil`.

The **handent** function returns entities that have been deleted during the current editing session. You can undelete them with the **entdel** function.

An entity's name can change from one editing session to the next, but an entity's handle remains constant.

Examples

```
Command: (handent "5A2")
<Entity name: 60004722>
```

Used with the same drawing but in another editing session, the same call might return a different entity name. Once the entity name is obtained, you can use it to manipulate the entity with any of the entity-related functions.

See also:

The entdel (page 67), entget (page 68), entlast (page 69), entmake (page 70), entmakex (page 72), entmod (page 73), <Undefined Cross-Reference> (page 75), entsel (page 77), and entupd (page 78) functions.

I Functions

if

Conditionally evaluates expressions

```
(if
  testexpr thenexpr [elseexpr]
)
```

Arguments

testexpr Expression to be tested.

thenexpr Expression evaluated if *testexpr* is not `nil`.

elseexpr Expression evaluated if *testexpr* is `nil`.

Return Values

The **if** function returns the value of the selected expression. If *elseexpr* is missing and *testexpr* is `nil`, then it returns `nil`.

Examples

```
Command: (if (= 1 3) "YES!!" "no.")
"no."
Command: (if (= 2 (+ 1 1)) "YES!!")
"YES!!"
Command: (if (= 2 (+ 3 4)) "YES!!")
nil
```

See also:

The progn (page 161) function.

initcommandversion

Forces the next command to run with the specified version.

```
(initcommandversion [version])
```

This function makes it possible to force a specific behavior for a supported command regardless of how it is being run. This only affects commands that have been updated to support a command version. In such commands, a test for an initialized command version replaces the legacy test for whether the command is being run from LISP or a script. When a supported command is being run manually, the default version is 2 (or the latest version). When a command is being run from automation, the default version is 1.

Arguments

version This argument specifies the version of the command to be used. If this argument is not present, the next use (and next use only) of a supported command will initialize to the latest version.

Return Values

`T`

Examples

Initializing a specific command version may affect each supported command differently. For example, here is the FILLET command with and without an initialized version:

```
Command: FILLET
```

```
Current settings: Mode = TRIM, Radius = 0.0000
Select first object or [Undo/Polyline/Radius/Trim/Multiple]:
*Cancel*

Command: (INITCOMMANDVERSION 1)

Command: FILLET
Current settings: Mode = TRIM, Radius = 0.0000
Select first object or [uNdo/Polyline/Radius/Trim/mUltiple]:
*Cancel*
```

Another typical example is the COLOR command. Run normally, COLOR displays the Select Color dialog; but by running (initcommandversion 1) before the COLOR command, it is forced to prompt from color from the command line.

initdia

Forces the display of the next command's dialog box

```
(initdia
   [dialogflag]
)
```

Currently, the following commands make use of the **initdia** function: ATTDEF, ATTEXT, BLOCK, COLOR, HATCH, IMAGE, IMAGEADJUST, INSERT, LAYER, LINETYPE, MTEXT, PLOT, RENAME, STYLE, and VIEW.

Arguments

dialogflag An integer. If this argument is not present or is present and nonzero, the next use (and next use only) of a command will display that command's dialog box rather than its command line prompts.

If *dialogflag* is zero, any previous call to this function is cleared, restoring the default behavior of presenting the command line interface.

Return Values

```
nil
```

Examples

Issue the PLOT command without calling **initdia** first:

```
Command: (command "_.PLOT")
plot
```

```
Detailed plot configuration? [Yes/No] <No>: nil
Detailed plot configuration? [Yes/No] <No>:
```

AutoCAD prompts for user input in the command window.

Use the following sequence of function calls to make AutoCAD display the Plot dialog box:

```
(initdia)
(command "_.PLOT")
```

initget

Establishes keywords for use by the next user-input function call

```
(initget
  [bits] [string]
)
```

The functions that honor keywords are **getint**, **getreal**, **getdist**, **getangle**, **getorient**, **getpoint**, **getcorner**, **getkword**, **entsel**, **nentsel**, and **nentselp**. The **getstring** function is the only user-input function that does not honor keywords.

The keywords are checked by the next user-input function call when the user does not enter the expected type of input (for example, a point to **getpoint**). If the user input matches a keyword from the list, thefunction returns that keyword as a string result. The application can test for the keywords and perform the action associated with each one. If the user input is not an expected type and does not match a keyword, AutoCAD asks the user to try again. The **initget** bit values and keywords apply only to the next user-input function call.

If **initget** sets a control bit and the application calls a user-input function for which the bit has no meaning, the bit is ignored.

If the user input fails one or more of the specified conditions (as in a zero value when zero values are not allowed), AutoCAD displays a message and asks the user to try again.

Arguments

bits A bit-coded integer that allows or disallows certain types of user input. The bits can be added together in any combination to form a value between

0 and 255. If no *bits* argument is supplied, zero (no conditions) is assumed. The bit values are as follows:

1 (bit 0) Prevents the user from responding to the request by entering only ENTER.

2 (bit 1) Prevents the user from responding to the request by entering zero.

4 (bit 2) Prevents the user from responding to the request by entering a negative value.

8 (bit 3) Allows the user to enter a point outside the current drawing limits. This condition applies to the next user-input function even if the AutoCAD system variable LIMCHECK is currently set.

16 (bit 4) (Not currently used.)

32 (bit 5) Uses dashed lines when drawing a rubber-band line or box. For those functions with which the user can specify a point by selecting a location in the drawing area, this bit value causes the rubber-band line or box to be dashed instead of solid. (Some display drivers use a distinctive color instead of dashed lines.) If the system variable POPUPS is 0, AutoCAD ignores this bit.

64 (bit 6) Prohibits input of a *Z* coordinate to the **getdist** function; lets an application ensure that this function returns a 2D distance.

128 (bit 7) Allows arbitrary input as if it is a keyword, first honoring any other control bits and listed keywords. This bit takes precedence over bit 0; if bits 7 and 0 are set and the user presses ENTER, a null string is returned.

256 (bit 8) Give direct distance input precedence over arbitrary input. For external applications, arbitrary input is given precedence over direct distance input by default. Set this bit if you wish to force AutoCAD to evaluate user input as direct distance input. Note that legal point input from the keyboard always takes precedence over either direct distance or arbitrary input.

512 (bit 9) If set before a call to **getpoint** or **getcorner**, a temporary UCS will be established when the cursor crosses over the edge of a planar face of a solid. The temporary UCS is reset when the cursor moves off of a face. It is dynamically re-established when the cursor moves over a different face. After the point is acquired, the dynamic UCS is reset to the current UCS. This functionality is not enabled for non-planar faces such as the side of a cylinder.

1024 (bit 10) When calling **getdist**, **getangle**, **getorient**, **getpoint**, or **getcorner**, you may not want the distance, angle, orient, point, or corner to be influenced by ortho, polar, or otracking in the *Z* direction. Setting this bit before calls to any of these functions will temporarily disable ortho, polar, and otracking in the *Z* direction. This is useful when you create 2D entities such as PLINE, ARC, or CIRCLE, or when you use the ARRAY command, which creates only a 2D array. In 2D-only commands it can be confusing and

error-prone to allow 3D points to be entered using ortho *Z*, polar *Z*, or otrack *Z*.

NOTE Future versions of AutoLISP may use additional **initget** control bits, so avoid setting bits that are not listed here.

string A string representing a series of keywords. See "Keyword Specifications" for information on defining keywords.

Return Values

`nil`

Function Applicable Control Bits

The special control values are honored only by those **get*xxx*** functions for which they make sense, as indicated in the following table:

User-input functions and applicable control bits

		Control bits values				
Function	**Honors key words**	**No null (1)**	**No zero (2)**	**No negative (4)**	**No limits (8)**	**Uses dashes (32)**
`getint`	X	X	X	X		
`getreal`	X	X	X	X		
`getdist`	X	X	X	X		X
`getangle`	X	X	X			X
`getorient`	X	X	X			X
`getpoint`	X	X			X	X
`getcorner`	X	X			X	X
`getkword`	X	X				
`entsel`	X					

User-input functions and applicable control bits

		Control bits values				
Function	**Honors key words**	**No null (1)**	**No zero (2)**	**No negative (4)**	**No limits (8)**	**Uses dashes (32)**
`nentsel`	X					
`nentselp`	X					

User-input functions and applicable control bits (continued)

	Control bits values				
Function	**2D distance (64)**	**Arbitrary input (128)**	**Direct distance (256)**	**UCS face tracking (512)**	**Disable Z-tracking (1024)**
`getint`		X			
`getreal`		X			
`getdist`	X	X	X		X
`getangle`		X	X		X
`getorient`		X	X		X
`getpoint`		X	X	X	X
`getcorner`		X	X	X	X
`getkword`		X			
`entsel`					
`nentsel`					

User-input functions and applicable control bits (continued)

	Control bits values				
Function	**2D distance (64)**	**Arbitrary input (128)**	**Direct distance (256)**	**UCS face tracking (512)**	**Disable Z-tracking (1024)**
`nentselp`					

Keyword Specifications

The *string* argument is interpreted according to the following rules:

1. Each keyword is separated from the following keyword by one or more spaces. For example, `"Width Height Depth"` defines three keywords.
2. Each keyword can contain only letters, numbers, and hyphens (-).

There are two methods for abbreviating keywords:

- The required portion of the keyword is specified in uppercase characters, and the remainder of the keyword is specified in lowercase characters. The uppercase abbreviation can be anywhere in the keyword (for example, `"LType"`, `"eXit"`, or `"toP"`).
- The *entire* keyword is specified in uppercase characters, and it is followed immediately by a comma, which is followed by the required characters (for example, `"LTYPE,LT"`). The keyword characters in this case must include the first letter of the keyword, which means that `"EXIT,X"` is not valid.

The two brief examples, `"LType"` and `"LTYPE,LT"`, are equivalent: if the user types **LT** (in either uppercase or lowercase letters), this is sufficient to identify the keyword. The user can enter characters that *follow* the required portion of the keyword, provided they don't conflict with the specification. In the example, the user could also enter **LTY** or **LTYP**, but **L** would not be sufficient.

If *string* shows the keyword entirely in uppercase *or* lowercase characters with no comma followed by a required part, AutoCAD recognizes the keyword only if the user enters all of it.

The **initget** function provides support for localized keywords. The following syntax for the keyword string allows input of the localized keyword while it returns the language independent keyword:

"

```
  local1
  local2
  localn
  _indep1
  indep2
  indepn
"
```

where *local1* through *localn* are the localized keywords, and *indep1* through *indepn* are the language-independent keywords.

There must always be the same number of localized keywords as language-independent keywords, and the first language-independent keyword is prefixed by an underscore as shown in the following example:

```
(initget "Abc Def _Ghi Jkl")
(getkword "\nEnter an option (Abc/Def): ")
```

Entering **A** returns Ghi and entering **_J** returns Jkl.

See also:

The entsel (page 77), getangle (page 91), getcorner (page 94), getdist (page 95), getint (page 99), getkword (page 100), getorient (page 101), getpoint (page 102), getreal (page 104), getstring (page 105), nentsel (page 149), and nentselp (page 151) functions. The Control of User-Input Function Conditions topic in the *AutoLISP Developer's Guide*.

ispropertyreadonly

Returns the read-only state of an entity's property.

```
(ispropertyreadonly
  ename propertyname [or collectionName index name]
)
```

Arguments

ename Name of the entity being queried. The ename can refer to either a graphical or a non-graphical entity.

propertyname Name of the property being queried. For a list of all the valid property names of a given object, use `dumpallproperties`.

collectionName If the object is a collection object, the Collection name is passed here.

index The collection index being queried.

name The name of the property within the collection being queried.

Return Values

1 is returned when the property is read-only; otherwise, 0 is returned when the property is writable.

Examples

The following example demonstrates how to check the read-only state of the Radius and Area properties of a circle.

```
Command: (setq e1 (car (entsel "\nSelect an arc or circle: ")))
<Entity name: 10e2e4ba0>
Command: (ispropertyreadonly e1 "Radius")
0

Command: (ispropertyreadonly e1 "Area")
1
```

See also:

DumpAllProperties (page 64)
GetPropertyValue (page 103)
SetPropertyValue (page 175)

inters

Finds the intersection of two lines

```
(inters
  pt1 pt2 pt3 pt4 [onseg]
)
```

All points are expressed in terms of the current UCS. If all four point arguments are 3D, **inters** checks for 3D intersection. If any of the points are 2D, **inters** projects the lines onto the current construction plane and checks only for 2D intersection.

Arguments

pt1 One endpoint of the first line.

pt2 The other endpoint of the first line.

pt3 One endpoint of the second line.

pt4 The other endpoint of the second line.

onseg If specified as `nil`, the lines defined by the four *pt* arguments are considered infinite in length. If the *onseg* argument is omitted or is not `nil`, the intersection point must lie on both lines or **inters** returns `nil`.

Return Values

If the *onseg* argument is present and is `nil`, **inters** returns the point where the lines intersect, even if that point is off the end of one or both of the lines. If the *onseg* argument is omitted or is not `nil`, the intersection point must lie on both lines or **inters** returns `nil`. The **inters** function returns `nil` if the two lines do not intersect.

Examples

```
(setq a '(1.0 1.0) b '(9.0 9.0))
(setq c '(4.0 1.0) d '(4.0 2.0))
```

```
Command: (inters a b c d)
nil
Command: (inters a b c d T)
nil
Command: (inters a b c d nil)
(4.0 4.0)
```

itoa

Returns the conversion of an integer into a string

```
(itoa
  int
)
```

Arguments

int An integer.

Return Values

A string derived from int.

Examples

```
Command: (itoa 33)
"33"
Command: (itoa -17)
"-17"
```

See also:

The atoi (page 31) function.

L Functions

lambda

Defines an anonymous function

```
(lambda
  arguments expr
...)
```

Use the **lambda** function when the overhead of defining a new function is not justified. It also makes your intention more apparent by laying out the function at the spot where it is to be used. This function returns the value of its last *expr,* and is often used in conjunction with **apply** and/or **mapcar** to perform a function on a list.

Arguments

arguments Arguments passed to an expression.

expr An AutoLISP expression.

Return Values

Value of the last *expr*.

Examples

```
(apply '(lambda (x y z)
        (* x (- y z))
```

```
        )
        '(5 20 14)
    )
30
    (setq counter 0)
    (mapcar '(lambda (x)
          (setq counter (1+ counter))
          (* x 5)
        )
        '(2 4 -6 10.2)
    )
0 (10 20 -30 51.0)
```

last

Returns the last element in a list

```
(last
  lst
)
```

Arguments

lst A list.

Return Values

An atom or a list.

Examples

```
Command: (last '(a b c d e))
E
Command: (last '(a b c (d e)))
(D E)
```

layoutlist

Returns a list of all paper space layouts in the current drawing

```
(layoutlist)
```

Return Values

A list of strings.

Examples

```
Command: (layoutlist)
("Layout1" "Layout2")
```

layerstate-addlayers

Adds or updates a series of layers to a layer state

```
(layerstate-addlayers
  layerstatename (list layerstate layername state color
  linetype lineweight plotstyle)
)
```

Arguments

layerstatename A string specifying the name of the layer state to be updated.

layername A string specifying the name of the layer to be added or updated.

state An integer sum designating properties in the layer to be set.

1- Turns the layer off

2- Freeze the layer

4- Lock the layer

8- Flag the layer as No Plot

16- Set the layer as being frozen in new viewports

A nil value uses defaults of on, thawed, unlocked, plottable, and thawed in new viewports.

color A dotted pair specifying the layers color type and value, *e.g.* (62 . ColorIndex), (420 . TrueColor), or (430 . "colorbook$colorname").

linetype A string specifying the name of the layer linetype. The linetype must already be loaded in the drawing or the default of "Continuous" will be used. A nil value sets the layer linetype to "Continuous."

lineweight An integer corresponding to a valid lineweight, i.e., 35 = .35, 211 = 2.11. A nil value sets the layer lineweight to "Default."

plotstyle A string specifying the name of the layers plot style. The plotstyle name must already be loaded in the drawing or the default of "Normal" will be used. A nil value sets the layer plotstyle to "Normal." If the drawing is in color dependent mode, this setting is ignored.

Return Values

`T` if successful; otherwise `nil`

Examples

```
(layerstate-addlayers "myLayerState" (list "Walls" 4 '(62
 . 45) "Divide" 35 "10% Screen")
(list "Floors" 6 '(420 . 16235019) "Continuous" 40 "60%
Screen")
(list "Ceiling" 0 '(430 . "RAL CLASSIC$RAL 1003") "DOT"
nil nil)))
T
```

layerstate-compare

Compares a layerstate to the layers in the current drawing

```
(layerstate-compare
  layerstatename viewport
)
```

Arguments

layerstatename A string specifying the name of the layer state compare.

viewport An ename (ads_name) of the viewport to be used in the compare. If viewport is nil, the current viewport is used

Return Values

`T` if successful; otherwise `nil`

Examples

```
(layerstate-compare "myLayerState")
```

layerstate-delete

Deletes a layer state

```
(layerstate-delete
  layerstatename
)
```

Arguments

layerstatename A string specifying the name of the layer state to be deleted.

Return Values

`T` if the delete succeds; otherwise `nil`

Examples

```
(layerstate-delete "myLayerState")
T
```

layerstate-export

Exports a layer state to a specified file

```
(layerstate-export
  layerstatename filename
)
```

Arguments

layerstatename A string specifying the name of the layer to export.

filename A string specifying the name of the file to which the layer state should be exported.

Return Values

`T` if the export is successful;`nil` otherwise.

Examples

```
(layerstate-export "myLayerState" "/mylayerstate.las")
T
```

layerstate-getlastrestored

Returns the name of the last restored layer state in the current drawing

```
(layerstate-getlastrestored)
```

Return Values

Returns the name of the last restored layer state in the current drawing.

Examples

```
(layerstate-getlastrestored)
"Foundation"
```

layerstate-getlayers

Returns the layers saved in a layer state

```
(layerstate-getlayers
  layerstatename [invert]
)
```

Arguments

layerstatename A string specifying the name of the layer state to query for layers.

invert If invert is omitted or `nil`, returns a list of the layers saved in the layer state. If invert is `T`, it returns a list of the layers in the drawing that are not saved in the layer state.

Return Values

A list of layer names. Returns `nil` if the layer state does not exist or contains no layers.

Examples

```
(layerstate-getlayers "myLayerState")
("Layername1" "Layername2")
```

layerstate-getnames

Returns a list of the layer state names

```
(layerstate-getnames [includehidden] [includexref])
```

Arguments

includehidden If *includehidden* is T, the list will include the names of hidden layer states. If omitted or nil, hidden layer states will be excluded.

includexref If *includexref* is nil, the list will exclude the names of xref layer states. If omitted or T, xref layer states will be included

Return Values

Returns a list of the layer state names

Examples

```
(layerstate-getnames)
("First Floor" "Second Floor" "Foundation")
```

layerstate-has

Checks if a layer state is present

```
(layerstate-has
  layerstatename
)
```

Arguments

layerstatename A string specifying the name of the layer state to be queried.

Return Values

`T` if the name exists; otherwise `nil`

Examples

```
(layerstate-has "myLayerState")
T
```

layerstate-import

Imports a layer state from a specified file

```
(layerstate-import
  filename
)
```

Arguments

filename A string specifying the name of the file from which to import a layer state.

Return Values

`T` if the import is successful; `nil` otherwise.

Examples

```
(layerstate-import "/mylayerstate.las")
T
```

layerstate-importfromdb

Imports a layer state from a specified drawing file

```
(layerstate-importfromdb
  layerstatename filename
)
```

Arguments

layerstatename A string specifying the name of the layer state to be imported.

filename A string specifying the name of the file from which to import a layer state.

Return Values

`T` if the import is successful; `nil` otherwise.

Examples

```
(layerstate-importfromdb "mylayerstate" "/mydrawing.dwg")
T
```

layerstate-removelayers

Removes a list of layers from a layer state

```
(layerstate-removelayers
  layerstatename (list layername layername layername ...)
)
```

Arguments

layerstatename A string specifying the name of the layer state to be updated.

layername A string specifying the name of the layer state to be removed.

Return Values

`T` if the remove is successful; otherwise `nil`

Examples

```
(layerstate-removelayers "myLayerState" (list "Walls"
"Elec1" "Foundation" "Plumbing"))
T
```

layerstate-rename

Renames a layer state

```
(layerstate-rename
  oldlayerstatename newlayerstatename
)
```

Arguments

oldlayerstatename A string specifying the name of the layer state to be renamed.

newlayerstatename A string specifying the name of the layer state to be updated.

Return Values

`T` if the rename is successful; otherwise `nil`

Examples

```
(layerstate-rename "myLayerState" "myNewLayerState")
T
```

layerstate-restore

Restores a layer state into the current drawing

```
(layerstate-restore
  layerstatename viewport [restoreflags]
)
```

Arguments

layerstatename A string specifying the name of the layer to restore.

viewport An ename (ads_name) of the viewport to which `layerstatename` should be restored. If viewport is nil, the layer state is restored to model space.

restoreflags Optional integer sum affecting how the layer state is restored.

1- Turn off all layers not in the restored layer state

2- Freeze all layers not in the restored layer state

4- Restore the layer state properties as viewport overrides (requires viewport to be not a `nil` value).

Return Values

`nil` if the layer state does not exist or contains no layers; otherwise, returns a list of layer names.

Examples

```
(layerstate-restore "myLayerState" viewportId 5)
("Layername1" "Layername2")
```

layerstate-save

Saves a layer state in the current drawing

```
(layerstate-save
  layerstatename mask viewport
)
```

Arguments

layerstatename A string specifying the name of the layer state to save.

mask An integer sum designating which properties in the layer state are to be restored.

1- Restore the saved On or Off value

2- Restore the saved Frozen or Thawed value

4- Restore the saved Lock value

8- Restore the saved Plot or No Plot value

16- Restore the saved VPVSDFLT value

32- Restore the saved Color

64- Restore the saved LineType

128- Restore the saved LineWeight

viewport An ename (ads_name) of the viewport whose VPLAYER setting is to be captured. If nil, the layer state will be saved without VPLAYER settings.

Return Values

`T` if the save is successful; otherwise `nil`

Examples

```
(layerstate-save "myLayerState" 21 viewportId)
T
(layerstate-save "myLayerState" nil nil)
nil
```

length

Returns an integer indicating the number of elements in a list

```
(length
  lst
)
```

Arguments

lst A list.

Return Values

An integer.

Examples

```
Command: (length '(a b c d))
4
Command: (length '(a b (c d)))
3
Command: (length '())
0
```

See also:

The vl-list-length (page 237) function.

list

Takes any number of expressions and combines them into one list

```
(list
  [expr
...
  ]
)
```

This function is frequently used to define a 2D or 3D point variable (a list of two or three reals).

Arguments

expr An AutoLISP expression.

Return Values

A list, unless no expressions are supplied, in which case **list** returns `nil`.

Examples

```
(list 'a 'b 'c)
(A B C)
```

(list 'a '(b c) 'd)

```
(A (B C) D)
```

(list 3.9 6.7)

```
(3.9 6.7)
```

As an alternative to using the **list** function, you can explicitly quote a list with the **quote** function if there are no variables or undefined items in the list. The single quote character (') is defined as the quote function.

'(3.9 6.7)

means the same as

```
(list 3.9 6.7)
```

This can be useful for creating association lists and defining points.

See also:

The quote (page 163), vl-list* (page 235), and vl-list-length (page 237) functions.

listp

Verifies that an item is a list

```
(listp
  item
)
```

Arguments

item Any atom, list, or expression.

Return Values

T if *item* is a list; otherwise nil. Because nil is both an atom and a list, the **listp** function returns T when passed nil.

Examples

```
Command: (listp '(a b c))
T
Command: (listp 'a)
nil
Command: (listp 4.343)
```

```
nil
Command: (listp nil)
T
Command: (listp (setq v1 '(1 2 43)))
T
```

See also:

The vl-list* (page 235) and vl-list-length (page 237) functions.

load

Evaluates the AutoLISP expressions in a file

```
(load
  filename [onfailure]
)
```

The **load** function can be used from within another AutoLISP function, or even recursively (in the file being loaded).

Arguments

filename A string that represents the file name. If the `filename` argument does not specify a file extension, **load** adds an extension to the name when searching for a file to load. The function will try several extensions, if necessary, in the following order:

As soon as **load** finds a match, it stops searching and loads the file.

The `filename` can include a directory prefix, as in */function/test1*. If you don't include a directory prefix in the `filename` string, **load** searches the AutoCAD library path for the specified file. If the file is found anywhere on this path, **load** then loads the file.

NOTE Use a single forward slash (`/`) or two backslashes (`\\`) as directory delimiters.

onfailure A value returned if **load** fails.

If the **onfailure** argument is a valid AutoLISP function, it is evaluated. In most cases, the **onfailure** argument should be a string or an atom. This allows an AutoLISP application calling **load** to take alternative action upon failure.

Return Values

Unspecified, if successful. If **load** fails, it returns the value of **onfailure**; if **onfailure** is not defined, failure results in an error message.

Examples

For the following examples, assume that file */fred/test1.lsp* contains the expressions

```
(defun MY-FUNC1 (x)
            ...
  function body
  ...
 ) (defun MY-FUNC2 (x)
            ...
  function body
  ...
```

and that no file named *test2* with a *.lsp* or *.fas* extension exists:

```
Command: (load "/fred/test1")
MY-FUNC2
Command: (load "/fred/test1" "bad")
MY-FUNC2
Command: (load "test2" "bad")
"bad"
Command: (load "test2") causes an AutoLISP error
```

See also:

The defun (page 52) and vl-load-all (page 238) functions. The Symbol and Function Handling topic in the *AutoLISP Developer's Guide*.

log

Returns the natural log of a number as a real number

```
(log
  num
)
```

Arguments

num A positive number.

Return Values

A real number.

Examples

```
Command: (log 4.5)
1.50408
Command: (log 1.22)
0.198851
```

logand

Returns the result of the logical bitwise AND of a list of integers

```
(logand
  [int int
...
  ]
)
```

Arguments

int An integer.

Return Values

An integer (0, if no arguments are supplied).

Examples

```
Command: (logand 7 15 3)
3
Command: (logand 2 3 15)
2
Command: (logand 8 3 4)
0
```

logior

Returns the result of the logical bitwise inclusive OR of a list of integers

```
(logior
  [int
  int
...
  ]
)
```

Arguments

int An integer.

Return Values

An integer (0, if no arguments are supplied).

Examples

```
Command: (logior 1 2 4)
7
Command: (logior 9 3)
11
```

lsh

Returns the logical bitwise shift of an integer by a specified number of bits

```
(lsh
  int
  numbits
)
```

Arguments

int An integer.

numbits Number of bits to shift *int*.

If *numbits* is positive, *int* is shifted to the left; if *numbits* is negative, *int* is shifted to the right. In either case, zero bits are shifted in, and the bits shifted out are discarded.

If *numbits* is not specified, no shift occurs.

Return Values

The value of *int* after the bitwise shift. The returned value is positive if the significant bit (bit number 31) contains a 0 after the shift operation; otherwise it is negative. If no arguments are supplied, **lsh** returns 0.

The behavior is different from other languages (>> & << of C, C++, or Java) where more than 32 left shifts (of a 32 bit integer) result in 0. In right shift, the integer appears again on every 32 shifts.

Examples

```
Command: (lsh 2 1)
4
Command: (lsh 2 -1)
1
Command: (lsh 40 2)
160
```

M Functions

mapcar

Returns a list that is the result of executing a function with a list (or lists) supplied as arguments to the function

```
(mapcar
  function
  list1
...
  listn
)
```

Arguments

function A function.

list1... listn One or more lists. The number of lists must match the number of arguments required by *function*.

Return Values

A list.

Examples

```
Command: (setq a 10 b 20 c 30)
30
Command: (mapcar '1+ (list a b c))
(11 21 31)
```

This is equivalent to the following series of expressions, except that **mapcar** returns a list of the results:

```
(1+ a)
(1+ b)
(1+ c)
```

The **lambda** function can specify an anonymous function to be performed by **mapcar**. This is useful when some of the function arguments are constant or are supplied by some other means. The following example, demonstrates the use of **lambda** with **mapcar**:

```
(mapcar  '(lambda (x)
          (+ x 3)
          )
          '(10 20 30)
)
(13 23 33)
```

max

Returns the largest of the numbers given

```
(max [
  number number
...])
```

Arguments

number A number.

Return Values

A number. If any of the arguments are real numbers, a real is returned; otherwise an integer is returned. If no argument is supplied, **max** returns 0.

Examples

```
Command: (max 4.07 -144)
4.07
Command: (max -88 19 5 2)
19
Command: (max 2.1 4 8)
8.0
```

mem

Displays the current state of the AutoLISP memory

```
(mem)
```

The **mem** function displays statistics on AutoLISP memory usage. The first line of this statistics report contains the following information:

GC calls Number of garbage collection calls since AutoLISP started.

GC run time Total time spent collecting garbage (in milliseconds).

LISP objects are allocated in dynamic (heap) memory that is organized in segments and divided into pages. Memory is described under the heading, "Dynamic Memory Segments Statistics":

PgSz Dynamic memory page size (in KB).

Used Number of pages used.

Free Number of free (empty) pages.

FMCL Largest contiguous area of free pages.

Segs Number of segments allocated.

Type Internal description of the types of objects allocated in this segment. These include

lisp stacks—LISP internal stacks

bytecode area—compiled code function modules

CONS memory—CONS objects

::new—untyped memory requests served using this segment

DM Str—dynamic string bodies

DMxx memory—all other LISP nodes

bstack body—internal structure used for IO operations

The final line in the report lists the minimal segment size and the number of allocated segments. AutoLISP keeps a list of no more than three free segments in order to save system calls for memory requests.

All heap memory is global; that is, all AutoCAD documents share the same heap. This could change in future releases of AutoCAD.

Note that **mem** does not list all memory requested from the operating system; it lists only those requests served by the AutoLISP Dynamic Memory (DM) subsystem. Some AutoLISP classes do not use DM for memory allocation.

Return Values

```
nil
```

Examples

```
Command: (mem)

; GC calls: 23; GC run time: 298 ms
Dynamic memory segments statistic:
PgSz  Used  Free  FMCL  Segs  Type
 512    79    48    48     1  lisp stacks
 256  3706   423   142    16  bytecode area
4096   320    10    10    22  CONS memory
  32   769  1213  1089     1  ::new
4096   168    12    10    12  DM Str
4096   222     4     4    15  DMxx memory
 128     4   507   507     1  bstack body
Segment size: 65536, total used: 68, free: 0
nil
```

member

Searches a list for an occurrence of an expression and returns the remainder of the list, starting with the first occurrence of the expression

```
(member
  expr lst
)
```

Arguments

expr The expression to be searched for.

lst The list in which to search for *expr*.

Return Values

A list; otherwise `nil`, if there is no occurrence of *expr* in *lst*.

Examples

```
Command: (member 'c '(a b c d e))
(C D E)
Command: (member 'q '(a b c d e))
nil
```

menucmd

Evaluates DIESEL expressions

```
(menucmd
  string
)
```

The **menucmd** function also allows AutoLISP programs to take advantage of the DIESEL string expression language. Some things can be done more easily with DIESEL than with the equivalent AutoLISP code. The following code returns a string containing the current day and date:

```
(menucmd "M=$(edtime,$(getvar,date),DDDD\",\" D MONTH
YYYY)")
  "Sunday, 16 July 1995"
```

See also:

The *Customization Guide* for more information on using AutoLISP to access menu label status, and for information on using DIESEL.

min

Returns the smallest of the numbers given

```
(min [
```

```
    number number
  ...])
```

Arguments

number A number.

Return Values

A number. If any *number* argument is a real, a real is returned; otherwise, an integer is returned. If no argument is supplied, **min** returns 0.

Examples

```
Command: (min 683 -10.0)
-10.0
Command: (min 73 2 48 5)
2
Command: (min 73.0 2 48 5)
2.0
Command: (min 2 4 6.7)
2.0
```

minusp

Verifies that a number is negative

```
(minusp
  num
)
```

Arguments

num A number.

Return Values

`T` if *number* is negative; otherwise `nil`.

Examples

```
Command: (minusp -1)
T
Command: (minusp -4.293)
T
Command: (minusp 830.2)
```

```
nil
```

N Functions

namedobjdict

Returns the entity name of the current drawing's named object dictionary, which is the root of all nongraphical objects in the drawing

```
(namedobjdict)
```

Using the name returned by this function and the dictionary access functions, an application can access the nongraphical objects in the drawing.

nentsel

Prompts the user to select an object (entity) by specifying a point, and provides access to the definition data contained within a complex object

```
(nentsel
  [msg]
)
```

The **nentsel** function prompts the user to select an object. The current Object Snap mode is ignored unless the user specifically requests it. To provide additional support at the Command prompt, **nentsel** honors keywords defined by a previous call to **initget**.

Arguments

msg A string to be displayed as a prompt. If the *msg* argument is omitted, the Select Object prompt is issued.

Return Values

When the selected object is not complex (that is, not a 3D polyline or block), **nentsel** returns the same information as **entsel**. However, if the selected object is a 3D polyline, **nentsel** returns a list containing the name of the subentity (vertex) and the pick point. This is similar to the list returned by **entsel**, except that the name of the selected vertex is returned instead of the polyline header.

The **nentsel** function always returns the starting vertex of the selected 3D polyline segment. Picking the third segment of the polyline, for example, returns the third vertex. The Seqend subentity is never returned by **nentsel** for a 3D polyline.

NOTE

A lightweight polyline (lwpolyline entity) is defined in the drawing database as a single entity; it does not contain subentities.

Selecting an attribute within a block reference returns the name of the attribute and the pick point. When the selected object is a component of a block reference other than an attribute, **nentsel** returns a list containing four elements.

The first element of the list returned from picking an object within a block is the selected entity's name.

The second element is a list containing the coordinates of the point used to pick the object.

The third element is called the Model to World Transformation Matrix. It is a list consisting of four sublists, each of which contains a set of coordinates. This matrix can be used to transform the entity definition data points from an internal coordinate system called the Model Coordinate System (MCS), to the World Coordinate System (WCS). The insertion point of the block that contains the selected entity defines the origin of the MCS. The orientation of the UCS when the block is created determines the direction of the MCS axes.

NOTE **nentsel** is the only AutoLISP function that uses a matrix of this type; the **nentselp** function returns a matrix similar to those used by other AutoLISP and ObjectARX functions.

The fourth element is a list containing the entity name of the block that contains the selected object. If the selected object is in a nested block (a block within a block), the list also contains the entity names of all blocks in which the selected object is nested, starting with the innermost block and continuing outward until the name of the block that was inserted in the drawing is reported.

For information about converting MCS coordinates to WCS, see the Entity Context and Coordinate Transform Data topic in Using AutoLISP to Manipulate AutoCAD Objects in the *AutoLISP Developer's Guide*.

Examples

Draw a 3D polyline with multiple line segments; then load and run the following function and select different segments of the line. Pick off the line and then pick the same segments again to see the subentity handle. Try it with a lightweight polyline to see the difference.

```
(defun c:subent ()
  (while
     (setq Ent (entsel "\nPick an entity: "))
     (print (strcat "Entity handle is: "
          (cdr (assoc 5 (entget (car Ent))))))
   )
   (while
      (setq Ent (nentsel "\nPick an entity or subEntity:
"))
      (print (strcat "Entity or subEntity handle is:  "
          (cdr (assoc 5 (entget (car Ent))))))
   )
  (prompt "\nDone.")
  (princ)
)
```

See also:

The entsel (page 77), initget (page 118),and nentselp (page 151) functions. The Entity Name Functions in the *AutoLISP Developer's Guide*.

nentselp

Provides similar functionality to that of the *nentsel* function without the need for user input

```
(nentselp
  [msg] [pt]
)
```

Arguments

msg A string to be displayed as a prompt. If the *msg* argument is omitted, the Select object prompt is issued.

pt A selection point. This allows object selection without user input.

Return Values

The **nentselp** function returns a 4×4 transformation matrix, defined as follows:

The first three columns of the matrix specify scaling and rotation. The fourth column is a translation vector.

The functions that use a matrix of this type treat a point as a column vector of dimension 4. The point is expressed in *homogeneous coordinates*, where the fourth element of the point vector is a *scale factor* that is normally set to 1.0. The final row of the matrix, the vector [$M_{30}M_{31}M_{32}M_{33}$], has the nominal value of [0 0 0 1]; it is currently ignored by the functions that use this matrix format.

See also:

The initget (page 118) and nentsel (page 149) functions.

not

Verifies that an item evaluates to nil

```
(not
  item
)
```

Typically, the **null** function is used for lists, and **not** is used for other data types along with some types of control functions.

Arguments

item An AutoLISP expression.

Return Values

`T` if *item* evaluates to `nil`; otherwise `nil`.

Examples

```
Command: (setq a 123 b "string" c nil)
nil
Command: (not a)
nil
Command: (not b)
nil
Command: (not c)
T
```

```
Command: (not '())
T
```

See also:

The null (page 153) function.

nth

Returns the nth element of a list

```
(nth
  n lst
)
```

Arguments

n The number of the element to return from the list (zero is the first element).

lst The list.

Return Values

The *n*th element of *lst*. If *n* is greater than the highest element number of *lst*, **nth** returns `nil`.

Examples

```
Command: (nth 3 '(a b c d e))
D
Command: (nth 0 '(a b c d e))
A
Command: (nth 5 '(a b c d e))
nil
```

null

Verifies that an item is bound to nil

```
(null
  item
)
```

Arguments

item An AutoLISP expression.

Return Values

`T` if *item* evaluates to `nil`; otherwise `nil`.

Examples

```
Command: (setq a 123 b "string" c nil)
nil
Command: (null a)
nil
Command: (null b)
nil
Command: (null c)
T
Command: (null '())
T
```

See also:

The not (page 152) function.

numberp

Verifies that an item is a real number or an integer

```
(numberp
  item
)
```

Arguments

item An AutoLISP expression.

Return Values

`T` if *item* evaluates to a real or an integer; otherwise `nil`.

Examples

```
Command: (setq a 123 b 'a)
A
Command: (numberp 4)
T
```

```
Command: (numberp 3.8348)
T
Command: (numberp "Howdy")
nil
Command: (numberp a)
T
Command: (numberp b)
nil
Command: (numberp (eval b))
T
```

O Functions

open

Opens a file for access by the AutoLISP I/O functions

```
(open
  filename mode
)
```

Arguments

filename A string that specifies the name and extension of the file to be opened. If you do not specify the full path name of the file, **open** assumes you are referring to the AutoCAD default drawing directory.

mode Indicates whether the file is open for reading, writing, or appending. Specify a string containing one of the following letters:

r Open for reading.

w Open for writing. If *filename* does not exist, a new file is created and opened. If *filename* already exists, its existing data is overwritten. Data passed to an open file is not actually written until the file is closed with the **close** function.

a Open for appending. If *filename* does not exist, a new file is created and opened. If *filename* already exists, it is opened and the pointer is positioned at the end of the existing data, so new data you write to the file is appended to the existing data.

The *mode* argument can be uppercase or lowercase. Note that in releases prior to AutoCAD 2000, *mode* had to be specified in lowercase.

Return Values

If successful, **open** returns a file descriptor that can be used by the other I/O functions. If mode `"r"` is specified and *filename* does not exist, **open** returns `nil`.

Examples

Open an existing file:

```
Command: (setq a (open "/myutilities/help/filelist.txt" "r"))
#<file "/myutilities/help/filelist.txt">
```

The following examples issue **open** against files that do not exist:

```
Command: (setq f (open "/documents/new.tst" "w"))
#<file "/my documents/new.tst">
Command: (setq f (open "nosuch.fil" "r"))
nil
Command: (setq f (open "logfile" "a"))
#<file "logfile">
```

or

Returns the logical OR of a list of expressions

```
(or
  [expr
...
  ]
)
```

The **or** function evaluates the expressions from left to right, looking for a non-`nil` expression.

Arguments

expr The expressions to be evaluated.

Return Values

`T`, if a non-`nil` expression is found; otherwise `nil`, if all of the expressions are `nil` or no arguments are supplied.

Note that **or** accepts an atom as an argument and returns `T` if one is supplied.

Examples

```
Command: (or nil 45 '())
T
Command: (or nil '())
nil
```

osnap

Returns a 3D point that is the result of applying an Object Snap mode to a specified point

```
(osnap
  pt mode
)
```

Arguments

pt A point.

mode A string that consists of one or more valid Object Snap identifiers, such as `mid`, `cen`, and so on, separated by commas.

Return Values

A point; otherwise `nil`, if the pick did not return an object (for example, if there is no geometry under the pick aperture, or if the geometry is not applicable to the selected object snap mode). The point returned by **osnap** depends on the current 3D view, the AutoCAD entity around *pt*, and the setting of the APERTURE system variable in the *Command Reference*.

Examples

```
Command: (setq pt1 (getpoint))
(11.8637 3.28269 0.0)
Command: (setq pt2 (osnap pt1 "_end,_int"))
(12.1424 3.42181 0.0)
```

P Functions

polar

Returns the UCS 3D point at a specified angle and distance from a point

```
(polar
  pt ang dist
)
```

Arguments

pt A 2D or 3D point.

ang An angle expressed in radians relative to the world *X* axis. Angles increase in the counterclockwise direction, independent of the current construction plane.

dist Distance from the specified *pt*.

Return Values

A 2D or 3D point, depending on the type of point specified by *pt*.

Examples

Supplying a 3D point to **polar**:

```
Command: (polar '(1 1 3.5) 0.785398 1.414214)
(2.0 2.0 3.5)
```

Supplying a 2D point to **polar**:

```
Command: (polar '(1 1) 0.785398 1.414214)
(2.0 2.0)
```

prin1

Prints an expression to the command line or writes an expression to an open file

```
(prin1
  [expr [file-desc]]
)
```

Arguments

expr A string or AutoLISP expression. Only the specified *expr* is printed; no newline or space is included.

file-desc A file descriptor for a file opened for writing.

Return Values

The value of the evaluated *expr*. If called with no arguments, **prin1** returns a null symbol.

Used as the last expression in a function, **prin1** without arguments prints a blank line when the function completes, allowing the function to exit "quietly."

Examples

```
Command: (setq a 123 b '(a))
(A)
Command: (prin1 'a)
AA
```

The previous command printed A and returned A.

```
Command: (prin1 a)
123123
```

The previous command printed 123 and returned 123.

```
Command: (prin1 b)
(A)(A)
```

The previous command printed (A) and returned (A).

Each preceding example is displayed on the screen because no *file-desc* was specified. Assuming that `f` is a valid file descriptor for a file opened for writing, the following function call writes a string to that file and returns the string:

```
Command: (prin1 "Hello" f)
"Hello"
```

If *expr* is a string containing control characters, **prin1** expands these characters with a leading \, as shown in the following table:

Control codes

Code	Description
`\\`	\ character
`\"`	" character
`\e`	Escape character
`\n`	Newline character

Control codes	
Code	**Description**
\r	Return character
\t	TAB character
nnn	Character whose octal code is *nnn*

The following example shows how to use control characters:

```
Command: (prin1 (chr 2))
"\002""\002"
```

See also:

Displaying Messages in the *AutoLISP Developer's Guide*.

princ

Prints an expression to the command line, or writes an expression to an open file

```
(princ
   [expr [file-desc]]
)
```

This function is the same as **prin1**, except control characters in *expr* are printed without expansion. In general, **prin1** is designed to print expressions in a way that is compatible with **load**, while **princ** prints them in a way that is readable by functions such as **read-line**.

Arguments

expr A string or AutoLISP expression. Only the specified *expr* is printed; no newline or space is included.

file-desc A file descriptor for a file opened for writing.

Return Values

The value of the evaluated *expr*. If called with no arguments, **princ** returns a null symbol.

See also:

The Displaying Messages topic in the *AutoLISP Developer's Guide*.

print

Prints an expression to the command line, or writes an expression to an open file

```
(print
  [expr [file-desc]]
)
```

This function is the same as **prin1**, except it prints a newline character before *expr*, and prints a space following *expr*.

Arguments

expr A string or AutoLISP expression. Only the specified *expr* is printed; no newline or space is included.

file-desc A file descriptor for a file opened for writing.

Return Values

The value of the evaluated *expr*. If called with no arguments, **print** returns a null symbol.

See also:

The Displaying Messages topic in the *AutoLISP Developer's Guide*.

progn

Evaluates each expression sequentially and returns the value of the last expression

```
(progn
  [expr]
```

```
...)
```

You can use **progn** to evaluate several expressions where only one expression is expected.

Arguments

expr One or more AutoLISP expressions.

Return Values

The result of the last evaluated expression.

Examples

The **if** function normally evaluates one *then* expression if the test expression evaluates to anything but `nil`. The following example uses **progn** to evaluate two expressions following **if**:

```
(if (= a b)
  (progn
    (princ "\nA = B ")
    (setq a (+ a 10) b (- b 10))
  )
)
```

See also:

The if (page 115) function.

prompt

Displays a string on your screen's prompt area

```
(prompt
  msg
)
```

Arguments

msg A string.

Return Values

`nil`

Examples

```
Command: (prompt "New value: ")
New value: nil
```

See also:

The Displaying Messages topic in the *AutoLISP Developer's Guide*.

Q Functions

quit

Forces the current application to quit

```
(quit)
```

If **quit** is called, it returns the error message quit/exit abort and returns to the AutoCAD Command prompt.

See also:

The exit (page 84) function.

quote

Returns an expression without evaluating it

```
(quote
  expr
)
```

Arguments

expr An AutoLISP expression.

Return Values

The *expr* argument.

Examples

```
Command: (quote a)
```

```
A
```

The previous expression can also be written as '**a**. For example:

```
Command: !'a
A
Command: (quote (a b))
(A B)
```

See also:

The function (page 89) function.

R Functions

read

Returns the first list or atom obtained from a string

```
(read
  [string]
)
```

The **read** function parses the string representation of any LISP data and returns the first expression in the string, converting it to a corresponding data type.

Arguments

string A string. The *string* argument should not contain blanks, except within a list or string.

Return Values

A list or atom. The **read** function returns its argument converted into the corresponding data type. If no argument is specified, **read** returns `nil`.

If the string contains multiple LISP expressions separated by LISP symbol delimiters such as blanks, newline, tabs, or parentheses, only the first expression is returned.

Examples

```
Command: (read "hello")
HELLO
Command: (read "hello there")
```

```
HELLO
Command: (read "\"Hi Y'all\"")
"Hi Y'all"
Command: (read "(a b c)")
(A B C)
Command: (read "(a b c) (d)")
(A B C)
Command: (read "1.2300")
1.23
Command: (read "87")
87
Command: (read "87 3.2")
87
```

read-char

Returns the decimal ASCII code representing the character read from the keyboard input buffer or from an open file

```
(read-char
  [file-desc]
)
```

Arguments

file-desc A file descriptor (obtained from **open**) referring to an open file. If no *file-desc* is specified, **read-char** obtains input from the keyboard input buffer.

Return Values

An integer representing the ASCII code for a character. The **read-char** function returns a single newline character (ASCII code 10) whenever it detects an end-of-line character or character sequence.

Examples

The following example omits *file-desc*, so **read-char** looks for data in the keyboard buffer:

```
Command: (read-char)
```

The keyboard buffer is empty, so **read-char** waits for user input:

ABC

```
65
```

The user entered **ABC**; **read-char** returned the ASCII code representing the first character entered (A). The next three calls to **read-char** return the data remaining in the keyboard input buffer. This data translates to 66 (the ASCII code for the letter B), 67 (C), and 10 (newline), respectively:

```
Command: (read-char)
66
Command: (read-char)
67
Command: (read-char)
10
```

With the keyboard input buffer now empty, **read-char** waits for user input the next time it is called:

```
Command: (read-char)
```

read-line

Reads a string from the keyboard or from an open file, until an end-of-line marker is encountered

```
(read-line
  [file-desc]
)
```

Arguments

file-desc A file descriptor (obtained from **open**) referring to an open file. If no *file-desc* is specified, **read-line** obtains input from the keyboard input buffer.

Return Values

The string read by **read-line**, without the end-of-line marker. If **read-line** encounters the end of the file, it returns `nil`.

Examples

Open a file for reading:

```
Command: (setq f (open "/documents/new.txt" "r"))
#<file "/documents/new.txt">
```

Use **read-line** to read a line from the file:

```
Command: (read-line f)
"To boldly go where nomad has gone before."
```

Obtain a line of input from the user:

```
Command: (read-line)
To boldly go
"To boldly go"
```

redraw

Redraws the current viewport or a specified object (entity) in the current viewport

```
(redraw
  [ename [mode]]
)
```

If **redraw** is called with no arguments, the function redraws the current viewport. If called with an entity name argument, **redraw** redraws the specified entity.

The **redraw** function has no effect on highlighted or hidden entities; however, a REGEN command forces the entities to redisplay in their normal manner.

Arguments

ename The name of the entity name to be redrawn.

mode An integer value that controls the visibility and highlighting of the entity. The *mode* can be one of the following values:

1 Show entity

2 Hide entity (blank it out)

3 Highlight entity

4 Unhighlight entity

The use of entity highlighting (mode 3) must be balanced with entity unhighlighting (mode 4).

If *ename* is the header of a complex entity (a polyline or a block reference with attributes), **redraw** processes the main entity and all its subentities if the *mode* argument is positive. If the *mode* argument is negative, **redraw** operates on only the header entity.

Return Values

The **redraw** function always returns `nil`.

regapp

Registers an application name with the current AutoCAD drawing in preparation for using extended object data

```
(regapp
  application
)
```

Arguments

application A string naming the application. The name must be a valid symbol table name. See the description of <Undefined Cross-Reference> (page 180) for the rules AutoLISP uses to determine if a symbol name is valid.

Return Values

If an application of the same name has already been registered, this function returns `nil`; otherwise it returns the name of the application.

If registered successfully, the application name is entered into the APPID symbol table. This table maintains a list of the applications that are using extended data in the drawing.

Examples

```
(regapp "ADESK_4153322344")
(regapp "DESIGNER-v2.1-124753")
```

NOTE It is recommended that you pick a unique application name. One way of ensuring this is to adopt a naming scheme that uses the company or product name and a unique number (like your telephone number or the current date/time). The product version number can be included in the application name or stored by the application in a separate integer or real-number field; for example, `(1040 2.1)`.

rem

Divides the first number by the second, and returns the remainder

```
(rem
  [number number
...
  ]
)
```

Arguments

number Any number.

If you provide more than two numbers, **rem** divides the result of dividing the first number by the second with the third, and so on.

If you provide more than two numbers, **rem** evaluates the arguments from left to right. For example, if you supply three numbers, **rem** divides the first number by the second, then takes the result and divides it by the third number, returning the remainder of that operation.

Return Values

A number. If any *number* argument is a real, **rem** returns a real; otherwise, **rem** returns an integer. If no arguments are supplied, **rem** returns 0. If a single *number* argument is supplied, **rem** returns *number*.

Examples

```
Command: (rem 42 12)
6
Command: (rem 12.0 16)
12.0
Command: (rem 26 7 2)
1
```

repeat

Evaluates each expression a specified number of times, and returns the value of the last expression

```
(repeat
  int [expr
...
  ]
)
```

Arguments

int An integer. Must be a positive number.

expr One or more atoms or expressions.

Return Values

The value of the last expression or atom evaluated. If *expr* is not supplied, **repeat** returns `nil`.

Examples

```
Command: (setq a 10 b 100)
100
Command: (repeat 4 (setq a (+ a 10)) (setq b (+ b 100)))
500
```

After evaluation, `a` is 50, `b` is 500, and **repeat** returns 500.

If strings are supplied as arguments, **repeat** returns the last string:

```
Command: (repeat 100 "Me" "You")
"You"
```

reverse

Returns a copy of a list with its elements reversed

```
(reverse
  lst
)
```

Arguments

lst A list.

Return Values

A list.

Examples

```
Command: (reverse '((a) b c))
(C B (A))
```

rtos

Converts a number into a string

```
(rtos
  number [mode [precision]]
)
```

The **rtos** function returns a string that is the representation of *number* according to the settings of *mode*, *precision*, and the system variables UNITMODE, DIMZIN, LUNITS, and LUPREC.

Arguments

number A number.

mode An integer specifying the linear units mode. The *mode* corresponds to the values allowed for the LUNITS AutoCAD system variable. The mode can be one of the following numbers:

1 Scientific

2 Decimal

3 Engineering (feet and decimal inches)

4 Architectural (feet and fractional inches)

5 Fractional

precision An integer specifying the precision.

The *mode* and *precision* arguments correspond to the system variables LUNITS and LUPREC. If you omit the arguments, **rtos** uses the current settings of LUNITS and LUPREC.

Return Values

A string. The UNITMODE system variable affects the returned string when engineering, architectural, or fractional units are selected (*mode* values 3, 4, or 5).

Examples

Set variable `x`:

```
Command: (setq x 17.5)
17.5
```

Convert the value of `x` to a string in scientific format, with a precision of 4:

```
Command: (setq fmtval (rtos x 1 4))
```

```
"1.7500E+01"
```

Convert the value of x to a string in decimal format, with 2 decimal places:

```
Command: (setq fmtval (rtos x 2 2))
"17.50"
```

Convert the value of x to a string in engineering format, with a precision of 2:

```
Command: (setq fmtval (rtos x 3 2))
"1'-5.50\""
```

Convert the value of x to a string in architectural format:

```
Command: (setq fmtval (rtos x 4 2))
"1'-5 1/2\""
```

Convert the value of x to a string in fractional format:

```
Command: (setq fmtval (rtos x 5 2))
"17 1/2"
```

Setting UNITMODE to 1 causes units to be displayed as entered. This affects the values returned by **rtos** for engineering, architectural, and fractional formats, as shown in the following examples:

```
Command: (setvar "unitmode" 1)
1
Command: (setq fmtval (rtos x 3 2))
"1'5.50\""
Command: (setq fmtval (rtos x 4 2))
"1'5-1/2\""
Command: (setq fmtval (rtos x 5 2))
"17-1/2"
```

See also:

The String Conversions topic in the *AutoLISP Developer's Guide* .

S Functions

set

Sets the value of a quoted symbol name to an expression

```
(set
  sym expr
)
```

The **set** function is similar to **setq** except that **set** evaluates both of its arguments whereas **setq** only evaluates its second argument.

Arguments

sym A symbol.

expr An AutoLISP expression.

Return Values

The value of the expression.

Examples

Each of the following commands sets symbol `a` to 5.0:

```
(set 'a 5.0)
(set (read "a") 5.0)
(setq a 5.0)
```

Both **set** and **setq** expect a symbol as their first argument, but **set** accepts an expression that returns a symbol, whereas **setq** does not, as the following shows:

```
Command: (set (read "a") 5.0)
5.0
Command: (setq (read "a") 5.0)
; *** ERROR: syntax error
```

See also:

The setq (page 176) function.

setcfg

Writes application data to the AppData section of the *acad.cfg* file

```
(setcfg
  cfgname cfgval
```

```
)
```

Arguments

cfgname A string that specifies the section and parameter to set with the value of *cfgval*. The *cfgname* argument must be a string of the following form:

AppData/*application_name*/*section_name*/.../*param_name*

The string can be up to 496 characters long.

cfgval A string. The string can be up to 512 characters in length. Larger strings are accepted by **setcfg**, but cannot be returned by **getcfg**.

Return Values

If successful, **setcfg** returns *cfgval*. If *cfgname* is not valid, **setcfg** returns `nil`.

Examples

The following code sets the WallThk parameter in the AppData/ArchStuff section to 8, and returns the string "8":

```
Command: (setcfg "AppData/ArchStuff/WallThk" "8")
"8"
```

See also:

The getcfg (page 93) function.

setenv

Sets a system environment variable to a specified value

```
(setenv
  varname value
)
```

Arguments

varname A string specifying the name of the environment variable to be set. Environment variable names must be spelled and cased exactly as they are stored in the system registry.

value A string specifying the value to set *varname* to.

Return Values

value

Examples

The following command sets the value of the `MaxArray` environment variable to 10000:

```
Command: (setenv "MaxArray" "10000")
"10000"
```

Note that changes to settings might not take effect until the next time AutoCAD is started.

See also:

The getenv (page 96) function.

setpropertyvalue

Sets the property value for an entity.

```
(setpropertyvalue
  ename propertyname value [or collectionName index name
val]
)
```

Arguments

ename Name of the entity being modified. The ename can refer to either a graphical or a non-graphical entity.

propertyname Name of the property to be modified. For a list of all the valid property names of a given object, use `dumpallproperties`.

value Value to set the property to when the object is not a collection.

collectionName If the object is a collection object, the Collection name is passed here.

index The collection index to be modified.

name Name of the property in the collection to be modified.

val Value to set the property to.

Return Values

`nil` is returned unless an error occurs when the property value is being updated.

Examples

The following example demonstrates how to change the radius of a circle.

```
Command: (command "_circle" "2,2" 2)
nil
Command: (setpropertyvalue (entlast) "radius" 3)
```

The following example demonstrates how to apply overrides to a linear dimension.

```
Command: (command "_dimlinear" "2,2" "5,4" "3,3")
nil
Command: (setq e2 (entlast))
<Entity name: 10e2e4bd0>
Command: (setpropertyvalue e2 "Dimtfill" 2)
nil
Command: (setpropertyvalue e2 "Dimtfillclr" "2")
nil
Command: (setpropertyvalue e2 "Dimclrt" "255,0,0")
nil
```

The following example demonstrates how to change the first vertex of the Vertices collection.

```
Command: (command "_pline" "0,0" "3,3" "5,2" "")
nil
Command: (setq e3 (entlast))
<Entity name: 10e2e4da0>
Command: (setpropertyvalue e3 "Vertices" 0 "EndWidth" 1.0)
nil
```

See also:

DumpAllProperties (page 64)
GetPropertyValue (page 103)
IsPropertyReadOnly (page 123)

setq

Sets the value of a symbol or symbols to associated expressions

```
(setq
  sym expr [sym expr]
...)
```

This is the basic assignment function in AutoLISP. The **setq** function can assign multiple symbols in one call to the function.

Arguments

sym A symbol. This argument is not evaluated.

expr An expression.

Return Values

The result of the last *expr* evaluated.

Examples

The following function call sets variable `a` to 5.0:

```
Command: (setq a 5.0)
5.0
```

Whenever `a` is evaluated, it returns the real number 5.0.

The following command sets two variables, `b` and `c`:

```
Command: (setq b 123 c 4.7)
4.7
```

setq returns the value of the last variable set.

In the following example, `s` is set to a string:

```
Command: (setq s "it")
"it"
```

The following example assigns a list to `x`:

```
Command: (setq x '(a b))
(A B)
```

See also:

The AutoLISP Variables topic in the *AutoLISP Developer's Guide* .

setvar

Sets an AutoCAD system variable to a specified value

```
(setvar
  varname value
)
```

Arguments

varname A string or symbol naming a variable.

value An atom or expression whose evaluated result is to be assigned to *varname*. For system variables with integer values, the supplied *value* must be between -32,768 and +32,767.

Return Values

If successful, **setvar** returns *value*.

Examples

Set the AutoCAD fillet radius to 0.5 units:

```
Command: (setvar "FILLETRAD" 0.50)
0.5
```

Notes on Using setvar

Some AutoCAD commands obtain the values of system variables before issuing any prompts. If you use **setvar** to set a new value while a command is in progress, the new value might not take effect until the next AutoCAD command.

When using the **setvar** function to change the AutoCAD system variable ANGBASE, the *value* argument is interpreted as radians. This differs from the AutoCAD SETVAR command in the *Command Reference*, which interprets this argument as degrees. When using the **setvar** function to change the AutoCAD system variable SNAPANG, the *value* argument is interpreted as radians relative to the AutoCAD default direction for angle 0, which is *east* or *3 o'clock*. This also differs from the SETVAR command, which interprets this argument as degrees relative to the ANGBASE setting.

NOTE The UNDO command does not undo changes made to the CVPORT system variable by the **setvar** function.

You can find a list of the current AutoCAD system variables in the *Command Reference*.

See also:

The getvar (page 106) function.

setview

Establishes a view for a specified viewport

```
(setview
  view_descriptor [vport_id]
)
```

Arguments

view_descriptor An entity definition list similar to that returned by **tblsearch** when applied to the VIEW symbol table.

vport_id An integer identifying the viewport to receive the new view. If *vport_id* is 0, the current viewport receives the new view.
You can obtain the *vport_id* number from the CVPORT system variable.

Return Values

If successful, the **setview** function returns the *view_descriptor*.

sin

Returns the sine of an angle as a real number expressed in radians

```
(sin
  ang
)
```

Arguments

ang An angle, in radians.

Return Values

A real number representing the sine of *ang*, in radians.

Examples

```
Command: (sin 1.0)
0.841471
Command: (sin 0.0)
0.0
```

snvalid

Checks the symbol table name for valid characters

```
(snvalid
  sym_name [flag]
)
```

The **snvalid** function inspects the system variable EXTNAMES to determine the rules to enforce for the active drawing. If EXTNAMES is 0, **snvalid** validates using the symbol name rules in effect prior to AutoCAD 2000. If EXTNAMES is 1 (the default value), **snvalid** validates using the rules for extended symbol names introduced with AutoCAD 2000. The following are not allowed in symbol names, regardless of the setting of EXTNAMES:

- Control and graphic characters
- Null strings
- Vertical bars as the first or last character of the name

AutoLISP does not enforce restrictions on the length of symbol table names if EXTNAMES is 1.

Arguments

sym_name A string that specifies a symbol table name.

flag An integer that specifies whether the vertical bar character is allowed within *sym_name*. The *flag* argument can be one of the following:

0 Do not allow vertical bar characters anywhere in *sym_name*. This is the default.

1 Allow vertical bar characters in *sym_name*, as long as they are not the first or last characters in the name.

Return Values

`T`, if *sym_name* is a valid symbol table name; otherwise `nil`.

If EXTNAMES is 1, all characters are allowed except control and graphic characters and the following:

Characters disallowed in symbol table names	
< >	less-than and greater-than symbol

Characters disallowed in symbol table names	
/ \	forward slash and backslash
"	quotation mark
:	colon
?	question mark
*	asterisk
|	vertical bar
,	comma
=	equal sign
`	backquote
;	semicolon (ASCII 59)

A symbol table name may contain spaces.

If EXTNAMES is 0, symbol table names can consist of uppercase and lowercase alphabetic letters (e.g., A-Z), numeric digits (e.g., 0-9), and the dollar sign ($), underscore (_), and hyphen (-) characters.

Examples

The following examples assume EXTNAMES is set to 1:

```
Command: (snvalid "hocus-pocus")
T
Command: (snvalid "hocus pocus")
T
Command: (snvalid "hocus%pocus")
T
```

The following examples assume EXTNAMES is set to 0:

```
Command: (snvalid "hocus-pocus")
```

```
T
Command: (snvalid "hocus pocus")
nil
Command: (snvalid "hocus%pocus")
nil
```

The following example includes a vertical bar in the symbol table name:

```
Command: (snvalid "hocus|pocus")
nil
```

By default, the vertical bar character is considered invalid in all symbol table names.

In the following example, the *flag* argument is set to 1, so **snvalid** considers the vertical bar character to be valid in *sym_name*, as long as it is not the first or last character in the name:

```
Command: (snvalid "hocus|pocus" 1)
T
```

sqrt

Returns the square root of a number as a real number

```
(sqrt
  num
)
```

Arguments

num A number (integer or real).

Return Values

A real number.

Examples

```
Command: (sqrt 4)
2.0
Command: (sqrt 2.0)
1.41421
```

ssadd

Adds an object (entity) to a selection set, or creates a new selection set

```
(ssadd
  [ename
   [ss]]
)
```

Arguments

ename An entity name.

ss A selection set.

If called with no arguments, **ssadd** constructs a new selection set with no members. If called with the single entity name argument *ename*, **ssadd** constructs a new selection set containing that single entity. If called with an entity name and the selection set *ss*, **ssadd** adds the named entity to the selection set.

Return Values

The modified selection set passed as the second argument, if successful; otherwise `nil`.

Examples

When adding an entity to a set, the new entity is added to the existing set, and the set passed as *ss* is returned as the result. Thus, if the set is assigned to other variables, they also reflect the addition. If the named entity is already in the set, the **ssadd** operation is ignored and no error is reported.

Set `e1` to the name of the first entity in drawing:

```
Command: (setq e1 (entnext))
<Entity name: 1d62d60>
```

Set `ss` to a null selection set:

```
Command: (setq ss (ssadd))
<Selection set: 2>
```

The following command adds the `e1` entity to the selection set referenced by `ss`:

```
Command: (ssadd e1 ss)
<Selection set: 2>
```

Get the entity following e1:
Command: **(setq e2 (entnext e1))**
<Entity name: 1d62d68>

Add e2 to the ss entity:
Command: **(ssadd e2 ss)**
<Selection set: 2>

ssdel

Deletes an object (entity) from a selection set

```
(ssdel
  ename
  ss
)
```

Arguments

ename An entity name.

ss A selection set.

Return Values

The name of the selection set; otherwise nil, if the specified entity is not in the set.

Note that the entity is actually deleted from the existing selection set, as opposed to a new set being returned with the element deleted.

Examples

In the following examples, entity name e1 is a member of selection set ss, while entity name e3 is not a member of ss:
Command: **(ssdel e1 ss)**
<Selection set: 2>

Selection set ss is returned with entity e1 removed.
Command: **(ssdel e3 ss)**
nil

The function returns nil because e3 is not a member of selection set ss.

ssget

Creates a selection set from the selected object

```
(ssget
  [sel-method] [pt1 [pt2]] [pt-list] [filter-list]
)
```

Selection sets can contain objects from both paper and model space, but when the selection set is used in an operation, **ssget** filters out objects from the space not currently in effect. Selection sets returned by **ssget** contain main entities only (no attributes or polyline vertices).

Arguments

sel-method A string that specifies the object selection method. Valid selection methods are

C Crossing selection.

CP Cpolygon selection (all objects crossing and inside of the specified polygon).

F Fence selection.

I Implied selection (objects selected while PICKFIRST is in effect).

L Last visible object added to the database.

P Last selection set created.

W Window selection.

WP WPolygon (all objects within the specified polygon).

X Entire database. If you specify the `x` selection method and do not provide a *filter-list*, **ssget** selects all entities in the database, including entities on layers that are off, frozen, and out of the visible screen.

:E Everything within the cursor's object selection pickbox.

:N Call **ssnamex** for additional information on container blocks and transformation matrices for any entities selected during the **ssget** operation. This additional information is available only for entities selected through graphical selection methods such as Window, Crossing, and point picks.

Unlike the other object selection methods, **:N** may return multiple entities with the same entity name in the selection set. For example, if the user selects a subentity of a complex entity such as a BlockReference, PolygonMesh, or old style polyline, **ssget** looks at the subentity that is selected when determining if it has already been selected. However, **ssget** actually adds the main entity (BlockReference, PolygonMesh, and so on) to the selection set.

The result could be multiple entries with the same entity name in the selection set (each will have different subentity information for **ssnamex** to report).

:R Allows entities in a long transaction to be selected.

:S Allow single selection only.

:U Enables subentity selection. Cannot be combined with the duplicate (":D") or nested (":N") selection modes. In this mode, top-level entities are selected by default, but the user can attempt to select subentities by pressing the CTRL key while making the selection. This option is supported only with interactive selections, such as window, crossing, and polygon. It is not supported for all, filtered, or group selections.

pt1 A point relative to the selection.

pt2 A point relative to the selection.

pt-list A list of points.

filter-list An association list that specifies object properties. Objects that match the *filter-list* are added to the selection set.

If you omit all arguments, **ssget** prompts the user with the Select Objects prompt, allowing interactive construction of a selection set.

If you supply a point but do not specify an object selection method, AutoCAD assumes the user is selecting an object by picking a single point.

Return Values

The name of the created selection set if successful; otherwise `nil` if no objects were selected.

Notes on the Object Selection Methods

- When using the `:N` selection method, if the user selects a subentity of a complex entity such as a BlockReference, PolygonMesh, or old style polyline, **ssget** looks at the subentity that is selected when determining if it has already been selected. However, **ssget** actually adds the main entity (BlockReference, PolygonMesh, etc.) to the selection set. It is therefore possible to have multiple entries with the same entity name in the selection set (each will have different subentity information for **ssnamex** to report). Because the `:N` method does not guarantee that each entry will be unique, code that relies on uniqueness should not use selection sets created using this option.
- When using the `L` selection method in an MDI environment, you cannot always count on the last object drawn to remain visible. For example, if your application draws a line, and the user subsequently minimizes or

cascades the AutoCAD drawing window, the line may no longer be visible. If this occurs, **ssget** with the `"L"` option will return `nil`.

Examples

Prompt the user to select the objects to be placed in a selection set:

```
Command: (ssget)
<Selection set: 2>
```

Create a selection set of the object passing through (2,2):

```
Command: (ssget '(2 2))
nil
```

Create a selection set of the most recently selected objects:

```
Command: (ssget "_P")
<Selection set: 4>
```

Create a selection set of the objects crossing the box from (0,0) to (1,1):

```
Command: (ssget "_C" '(0 0) '(1 1))
<Selection set: b>
```

Create a selection set of the objects inside the window from (0,0):

```
Command: (ssget "_W" '(0 0) '(5 5))
<Selection set: d>
```

By specifying filters, you can obtain a selection set that includes all objects of a given type, on a given layer, or of a given color. The following example returns a selection set that consists only of blue lines that are part of the implied selection set (those objects selected while PICKFIRST is in effect):

```
Command: (ssget "_I" '((0 . "LINE") (62 . 5)))
<Selection set: 4>
```

The following examples of **ssget** require that a list of points be passed to the function. The `pt_list` variable cannot contain points that define zero-length segments.

Create a list of points:

```
Command: (setq pt_list '((1 1)(3 1)(5 2)(2 4)))
((1 1) (3 1) (5 2) (2 4))
```

Create a selection set of all objects crossing and inside the polygon defined by *pt_list*:

```
Command: (ssget "_CP" pt_list)
<Selection set: 13>
```

Create a selection set of all blue lines inside the polygon defined by *pt_list*:

```
Command: (ssget "_WP" pt_list '((0 . "LINE") (62 . 5)))
<Selection set: 8>
```

The selected objects are highlighted only when **ssget** is used with no arguments. Selection sets consume AutoCAD temporary file slots, so AutoLISP is not permitted to have more than 128 open at one time. If this limit is reached, AutoCAD cannot create any more selection sets and returns `nil` to all **ssget** calls. To close an unnecessary selection set variable, set it to `nil`.

A selection set variable can be passed to AutoCAD in response to any Select objects prompt at which selection by Last is valid. AutoCAD then selects all the objects in the selection set variable.

The current setting of Object Snap mode is ignored by **ssget** unless you specifically request it while you are in the function.

See also:

Selection Set Handling and Selection Set Filter Lists in the *AutoLISP Developer's Guide* .

ssgetfirst

Determines which objects are selected and gripped

```
(ssgetfirst)
```

Returns a list of two selection sets similar to those passed to **sssetfirst**. The first element in the list is always `nil` because AutoCAD no longer supports grips on unselected objects. The second element is a selection set of entities that are selected and gripped. Both elements of the list can be `nil`.

NOTE

Only entities from the current drawing's model space and paper space, not nongraphical objects or entities in other block definitions, can be analyzed by this function.

See also:

The ssget (page 185) and sssetfirst (page 194) functions.

sslength

Returns an integer containing the number of objects (entities) in a selection set

```
(sslength
  ss
)
```

Arguments

ss A selection set.

Return Values

An integer.

Examples

Add the last object to a new selection set:

```
Command: (setq sset (ssget "L"))
<Selection set: 8>
```

Use **sslength** to determine the number of objects in the new selection set:

```
Command: (sslength sset)
1
```

ssmemb

Tests whether an object (entity) is a member of a selection set

```
(ssmemb
  ename
  ss
)
```

Arguments

ename An entity name.

ss A selection set.

Return Values

If *ename* is a member of *ss*, **ssmemb** returns the entity name. If *ename* is not a member, **ssmemb** returns `nil`.

Examples

In the following examples, entity name `e2` is a member of selection set `ss`, while entity name `e1` is not a member of `ss`:

```
Command: (ssmemb e2 ss)
<Entity name: 1d62d68>
Command: (ssmemb e1 ss)
nil
```

ssname

Returns the object (entity) name of the indexed element of a selection set

```
(ssname
  ss
   index
)
```

Entity names in selection sets obtained with **ssget** are always names of main entities. Subentities (attributes and polyline vertices) are not returned. (The **entnext** function allows access to them.)

Arguments

ss A selection set.

index An integer (or real) indicating an element in a selection set. The first element in the set has an index of zero. To access entities beyond number 32,767 in a selection set, you must supply the *index* argument as a real.

Return Values

An entity name, if successful. If *index* is negative or greater than the highest-numbered entity in the selection set, **ssname** returns `nil`.

Examples

Get the name of the first entity in a selection set:

```
Command: (setq ent1 (ssname ss 0))
<Entity name: 1d62d68>
```

Get the name of the fourth entity in a selection set:

```
Command: (setq ent4 (ssname ss 3))
<Entity name: 1d62d90>
```

To access entities beyond the number 32,767 in a selection set, you must supply the *index* argument as a real, as in the following example:

```
(setq entx (ssname sset 50843.0))
```

See also:

The entnext (page 75) function.

ssnamex

Retrieves information about how a selection set was created

```
(ssnamex
  ss [index]
)
```

Only selection sets with entities from the current drawing's model space and paper space—*not* nongraphical objects or entities in other block definitions—can be retrieved by this function.

Arguments

ss A selection set.

index An integer (or real) indicating an element in a selection set. The first element in the set has an index of zero.

Return Values

If successful, **ssnamex** returns the name of the entity at *index*, along with data describing how the entity was selected. If the *index* argument is not supplied, this function returns a list containing the entity names of the elements in the selection set, along with data that describes how each entity was selected. If *index* is negative or greater than the highest-numbered entity in the selection set, **ssnamex** returns `nil`.

The data returned by **ssnamex** takes the form of a list of lists containing information that describes either an entity and its selection method or a polygon used to select one or more entities. Each sublist that describes the selection of a particular entity comprises three parts: the selection method ID

(an integer >= 0), the entity name of the selected entity, and selection method specific data that describes how the entity was selected.

```
((
  sel_id1 ename1
 (
  data
))(
  sel_id2
  ename2
 (
  data
)) ... )
```

The following table lists the selection method IDs:

Selection method IDs	
ID	**Description**
0	Nonspecific (i.e., Last All)
1	Pick
2	Window or WPolygon
3	Crossing or CPolygon
4	Fence

Each sublist that both describes a polygon and is used during entity selection takes the form of a polygon ID (an integer < 0), followed by point descriptions.

```
(
  polygon_id
  point_description_1
  point_description_n
... )
```

Polygon ID numbering starts at -1 and each additional polygon ID is incremented by -1. Depending on the viewing location, a point is represented as one of the following: an infinite line, a ray, or a line segment. A point descriptor comprises three parts: a point descriptor ID (the type of item being described), the start point of the item, and an optional unit vector that describes either the direction in which the infinite line travels or a vector that describes the offset to the other side of the line segment.

```
(
  point_descriptor_id
  base_point
  [unit_or_offset_vector]
)
```

The following table lists the valid point descriptor IDs:

Point descriptor IDs

ID	Description
0	Infinite line
1	Ray
2	Line segment

The *unit_or_offset_vector* is returned when the view point is something other than 0,0,1.

Examples

The *data* associated with Pick (type 1) entity selections is a single point description. For example, the following record is returned for the selection of an entity picked at 1,1 in plan view of the WCS:

```
Command: (ssnamex ss3 0)
((1 <Entity name: 1d62da0> 0 (0 (1.0 1.0 0.0))))
```

The *data* associated with an entity selected with the Window, WPolygon, Crossing, or CPolygon method is the integer ID of the polygon that selected the entity. It is up to the application to associate the polygon identifiers and make the connection between the polygon and the entities it selected. For example, the following returns an entity selected by Crossing (note that the polygon ID is -1):

```
Command: (ssnamex ss4 0)
((3 <Entity name: 1d62d60> 0 -1) (-1 (0 (-1.80879 8.85536
0.0)) (0 (13.4004 8.85536 0.0)) (0 (13.4004 1.80024 0.0)) (0
(-1.80879 1.80024 0.0))))
```

The data associated with fence selections is a list of points and descriptions for the points where the fence and entity visually intersect. For example, the following command returns information for a nearly vertical line intersected three times by a Z-shaped fence:

```
Command: (ssnamex ss5 0)
((4 <Entity name: 1d62d88> 0 (0 (5.28135 6.25219 0.0) ) (0
(5.61868 2.81961 0.0) ) (0 (5.52688 3.75381 0.0) ) ) )
```

sssetfirst

Sets which objects are selected and gripped

```
(sssetfirst
  gripset
  [pickset]
)
```

The *gripset* argument is ignored; the selection set of objects specified by *pickset* are selected and gripped.

You are responsible for creating a valid selection set. For example, you may need to verify that a background paper space viewport (DXF group code 69) is not included in the selection set. You may also need to ensure that selected objects belong to the current layout, as in the following code:

```
(setq ss (ssget (list (cons 410 (getvar "ctab")))))
```

Arguments

gripset AutoCAD no longer supports grips on unselected objects, so this argument is ignored. However, if *gripset* is `nil` and no *pickset* is specified, **sssetfirst** turns off the grip handles and selections it previously turned on.

pickset A selection set to be selected.

Return Values

The selection set or sets specified.

Examples

First, draw a square and build three selection sets. Begin by drawing side 1 and creating a selection set to include the line drawn:

```
Command: (entmake (list (cons 0 "line") '(10 0.0 0.0 0.0)'(11 0.0 10.0 0.0)))
((0 . "line") (10 0.0 0.0 0.0) (11 0.0 10.0 0.0))
Command: (setq pickset1 (ssget "_1"))
<Selection set: a5>
```

Variable `pickset1` points to the selection set created.

Draw side 2 and add it to the `pickset1` selection set:

```
Command: (entmake (list (cons 0 "line") '(10 0.0 10.0 0.0)'(11 10.0 10.0 0.0)))
((0 . "line") (10 0.0 10.0 0.0) (11 10.0 10.0 0.0))
Command: (ssadd (entlast) pickset1)
<Selection set: a5>
```

Create another selection set to include only side 2:

```
Command: (setq 2onlyset (ssget "_1"))
<Selection set: a8>
```

Draw side 3 and add it to the `pickset1` selection set:

```
Command: (entmake (list (cons 0 "line") '(10 10.0 10.0 0.0)'(11 10.0 0.0 0.0)))
((0 . "line") (10 10.0 10.0 0.0) (11 10.0 0.0 0.0))
Command: (ssadd (entlast) pickset1)
<Selection set: a5>
```

Create another selection and include side 3 in the selection set:

```
Command: (setq pickset2 (ssget "_1"))
<Selection set: ab>
```

Variable `pickset2` points to the new selection set.

Draw side 4 and add it to the `pickset1` and `pickset2` selection sets:

```
Command: (entmake (list (cons 0 "line") '(10 10.0 0.0 0.0)'(11 0.0 0.0 0.0)))
((0 . "line") (10 10.0 0.0 0.0) (11 0.0 0.0 0.0))
Command: (ssadd (entlast) pickset1)
<Selection set: a5>
Command: (ssadd (entlast) pickset2)
<Selection set: ab>
```

At this point, pickset1 contains sides 1-4, pickset2 contains sides 3 and 4, and 2onlyset contains only side 2.

Turn grip handles on and select all objects in pickset1:

```
Command: (sssetfirst nil pickset1)
(nil <Selection set: a5>)
```

Turn grip handles on and select all objects in pickset2:

```
Command: (sssetfirst nil pickset2)
(nil <Selection set: ab>)
```

Turn grip handles on and select all objects in 2onlyset:

```
Command: (sssetfirst nil 2onlyset)
(nil <Selection set: a8>)
```

Each **sssetfirst** call replaces the gripped and selected selection set from the previous **sssetfirst** call.

NOTE Do *not* call **sssetfirst** when AutoCAD is in the middle of executing a command.

See also:

The ssget (page 185) and ssgetfirst (page 188) functions.

startapp

Starts a Mac OS X application

```
(startapp
  appcmd
  [file]
)
```

Arguments

appcmd A string that specifies the application to execute. If *appcmd* does not include a full path name, **startapp** searches the directories in the PATH environment variable for the application.

file A string that specifies the file name to be opened.

Return Values

An integer greater than 0, if successful; otherwise `nil`.

Examples

The following code starts TextEdit and opens the *acad.lsp* file.

```
Command: (startapp "TextEdit.app" "acad.lsp")
33
```

The following code starts TextEdit and opens the *my stuff.txt* file in the */myutilities/lsupport* directory.

```
Command: (startapp "textedit.app" "/myutilities/support/my stuff.txt")
33
```

strcase

Returns a string where all alphabetic characters have been converted to uppercase or lowercase

```
(strcase
   string [which]
)
```

Arguments

string A string.

which If specified as `T`, all alphabetic characters in *string* are converted to lowercase. Otherwise, characters are converted to uppercase.

Return Values

A string.

Examples

```
Command: (strcase "Sample")
"SAMPLE"
Command: (strcase "Sample" T)
"sample"
```

The **strcase** function will correctly handle case mapping of the currently configured character set.

strcat

Returns a string that is the concatenation of multiple strings

```
(strcat
  [string
  [string]
...
  ]
)
```

Arguments

string A string.

Return Values

A string. If no arguments are supplied, **strcat** returns a zero-length string.

Examples

```
Command: (strcat "a" "bout")
"about"
Command: (strcat "a" "b" "c")
"abc"
Command: (strcat "a" "" "c")
"ac"
Command: (strcat)
""
```

strlen

Returns an integer that is the number of characters in a string

```
(strlen
  [string]
...)
```

Arguments

string A string.

Return Values

An integer. If multiple *string* arguments are provided, **strlen** returns the sum of the lengths of all arguments. If you omit the arguments or enter an empty string, **strlen** returns 0.

Examples

```
Command: (strlen "abcd")
4
Command: (strlen "ab")
2
Command: (strlen "one" "two" "four")
10
Command: (strlen)
0
Command: (strlen "")
0
```

subst

Searches a list for an old item and returns a copy of the list with a new item substituted in place of every occurrence of the old item

```
(subst
  newitem olditem lst
)
```

Arguments

newitem An atom or list.

olditem An atom or list.

lst A list.

Return Values

A list, with *newitem* replacing all occurrences of *olditem*. If *olditem* is not found in *lst*, **subst** returns *lst* unchanged.

Examples

```
Command: (setq sample '(a b (c d) b))
(A B (C D) B)
Command: (subst 'qq 'b sample)
(A QQ (C D) QQ)
```

```
Command: (subst 'qq 'z sample)
(A B (C D) B)
Command: (subst 'qq '(c d) sample)
(A B QQ B)
Command: (subst '(qq rr) '(c d) sample)
(A B (QQ RR) B)
Command: (subst '(qq rr) 'z sample)
(A B (C D) B)
```

When used in conjunction with **assoc**, **subst** provides a convenient means of replacing the value associated with one key in an association list, as demonstrated by the following function calls.

Set variable `who` to an association list:

```
Command: (setq who '((first john) (mid q) (last public)))
((FIRST JOHN) (MID Q) (LAST PUBLIC))
```

The following sets `old` to (FIRST JOHN) and `new` to (FIRST J):

```
Command: (setq old (assoc 'first who) new '(first j))
(FIRST J)
```

Finally, replace the value of the first item in the association list:

```
Command: (subst new old who)
((FIRST J) (MID Q) (LAST PUBLIC))
```

substr

Returns a substring of a string

```
(substr
  string start [length]
)
```

The **substr** function starts at the *start* character position of *string* and continues for *length* characters.

Arguments

string A string.

start A positive integer indicating the starting position in *string*. The first character in the string is position 1.

length A positive integer specifying the number of characters to search through in *string*. If *length* is not specified, the substring continues to the end of *string*.

NOTE The first character of *string* is character number 1. This differs from other functions that process elements of a list (like **nth** and **ssname**) that count the first element as 0.

Return Values

A string.

Examples

```
Command: (substr "abcde" 2)
"bcde"
Command: (substr "abcde" 2 1)
"b"
Command: (substr "abcde" 3 2)
"cd"
```

T Functions

tblnext

Finds the next item in a symbol table

```
(tblnext
  table-name [rewind]
)
```

When **tblnext** is used repeatedly, it normally returns the next entry in the specified table each time. The **tblsearch** function can set the *next* entry to be retrieved. If the *rewind* argument is present and is not `nil`, the symbol table is rewound and the first entry in it is retrieved.

Arguments

table-name A string that identifies a symbol table. Valid *table-name* values are `"LAYER"`, `"LTYPE"`, `"VIEW"`, `"STYLE"`, `"BLOCK"`, `"UCS"`, `"APPID"`, `"DIMSTYLE"`, and `"VPORT"`. The argument is not case sensitive.

rewind If this argument is present and is not `nil`, the symbol table is rewound and the first entry in it is retrieved.

Return Values

If a symbol table entry is found, the entry is returned as a list of dotted pairs of DXF-type codes and values. If there are no more entries in the table, `nil` is returned. Deleted table entries are never returned.

Examples

Retrieve the first layer in the symbol table:

```
Command: (tblnext "layer" T)
((0 . "LAYER") (2 . "0") (70 . 0) (62 . 7) (6 . "CONTINUOUS"))
```

The return values represent the following:

```
(0 . "LAYER")
  Symbol type

(2 . "0")
  Symbol name

(70 . 0)
  Flags

(62 . 7)
  Color number, negative if off

(6 . "CONTINUOUS")
  Linetype name
```

Note that there is no -1 group. The last entry returned from each table is stored, and the next one is returned each time **tblnext** is called for that table. When you begin scanning a table, be sure to supply a non-`nil` second argument to rewind the table and to return the first entry.

Entries retrieved from the block table include a -2 group with the entity name of the first entity in the block definition (if any). For example, the following command obtains information about a block called BOX:

```
Command: (tblnext "block")
((0 . "BLOCK") (2 . "BOX") (70 . 0) (10 9.0 2.0 0.0) (-2 .
<Entity name: 1dca370>))
```

The return values represent the following:

```
(0 . "BLOCK")
```

```
  Symbol type

(2 . "BOX")
  Symbol name

(70 . 0)
  Flags

(10 9.0 2.0 0.0)
  Origin X,Y,Z

(-2 . <Entity name: 1dca370>)
  First entity
```

The entity name in the -2 group is accepted by **entget** and **entnext**, but not by other entity access functions. For example, you cannot use **ssadd** to put it in a selection set. By providing the -2 group entity name to **entnext**, you can scan the entities comprising a block definition; **entnext** returns `nil` after the last entity in the block definition.

If a block contains no entities, the -2 group returned by **tblnext** is the entity name of its endblk entity.

NOTE The **vports** function returns current `VPORT` table information; therefore, it may be easier to use **vports** as opposed to **tblnext** to retrieve this information.

---

tblobjname

Returns the entity name of a specified symbol table entry

```
(tblobjname
  table-name symbol
)
```

Arguments

table-name A string that identifies the symbol table to be searched. The argument is not case-sensitive.

symbol A string identifying the symbol to be searched for.

Return Values

The entity name of the symbol table entry, if found.

The entity name returned by **tblobjname** can be used in **entget** and **entmod** operations.

Examples

The following command searches for the entity name of the block entry "ESC-01":

```
Command: (tblobjname "block" "ESC-01")
<Entity name: 1dca368>
```

tblsearch

Searches a symbol table for a symbol name

```
(tblsearch
  table-name symbol [setnext]
)
```

Arguments

table-name A string that identifies the symbol table to be searched. This argument is not case-sensitive.

symbol A string identifying the symbol name to be searched for. This argument is not case-sensitive.

setnext If this argument is supplied and is not `nil`, the **tblnext** entry counter is adjusted so the following **tblnext** call returns the entry after the one returned by this **tblsearch** call. Otherwise, **tblsearch** has no effect on the order of entries retrieved by **tblnext**.

Return Values

If **tblsearch** finds an entry for the given symbol name, it returns that entry in the format described for <Undefined Cross-Reference> (page 201). If no entry is found, **tblsearch** returns `nil`.

Examples

The following command searches for a text style named "standard":

```
Command: (tblsearch "style" "standard")
((0 . "STYLE") (2 . "STANDARD") (70 . 0) (40 . 0.0) (41 .
1.0) (50 . 0.0) (71 . 0) (42 . 0.3) (3 . "txt") (4 . ""))
```

terpri

Prints a newline to the command line

```
(terpri)
```

The **terpri** function is not used for file I/O. To write a newline to a file, use **prin1**, **princ**, or **print**.

Return Values

```
nil
```

textbox

Measures a specified text object, and returns the diagonal coordinates of a box that encloses the text

```
(textbox
   elist
)
```

Arguments

elist An entity definition list defining a text object, in the format returned by **entget**.

If fields that define text parameters other than the text itself are omitted from *elist*, the current (or default) settings are used.

The minimum list accepted by **textbox** is that of the text itself.

Return Values

A list of two points, if successful; otherwise `nil`.

The points returned by **textbox** describe the bounding box of the text object as if its insertion point is located at (0,0,0) and its rotation angle is 0. The first list returned is generally the point (0.0 0.0 0.0) unless the text object is oblique or vertical, or it contains letters with descenders (such as *g* and *p*). The value of the first point list specifies the offset from the text insertion point to the lower-left corner of the smallest rectangle enclosing the text. The second point list specifies the upper-right corner of that box. Regardless of the orientation

of the text being measured, the point list returned always describes the lower-left and upper-right corners of this bounding box.

Examples

The following command supplies the text and accepts the current defaults for the remaining parameters:

```
Command: (textbox '((1 . "Hello world.")))
((0.000124126 -0.00823364 0.0) (3.03623 0.310345 0.0))
```

textpage

Switches focus from the drawing area to the text screen

NOTE This function is supported on Mac OS, but does not affect AutoCAD.

```
(textpage)
```

The **textpage** function is equivalent to **textscr**.

Return Values

```
nil
```

textscr

Switches focus from the drawing area to the text screen

NOTE This function is supported on Mac OS, but does not affect AutoCAD.

```
(textscr)
```

Return Values

The **textscr** function always returns `nil`.

See also:

The graphscr (page 106) function.

trace

Aids in AutoLISP debugging

```
(trace
  [function
...
  ]
)
```

The **trace** function sets the trace flag for the specified functions. Each time a specified function is evaluated, a trace display appears showing the entry of the function (indented to the level of calling depth) and prints the result of the function.

Use **untrace** to turn off the trace flag.

Arguments

function A symbol that names a function. If no argument is supplied, **trace** has no effect.

Return Values

The last function name passed to **trace**. If no argument is supplied, **trace** returns `nil`.

Examples

Define a function named **foo** and set the trace flag for the function:

```
Command: (defun foo (x) (if (> x 0) (foo (1- x))))
FOO
Command: (trace foo)
FOO
```

Invoke **foo** and observe the results:

```
Command: (foo 3)
Entering (FOO 3)
Entering (FOO 2)
Entering (FOO 1)
Entering (FOO 0)
Result: nil
Result: nil
Result: nil
Result: nil
```

Clear the trace flag by invoking **untrace**:

```
Command: (untrace foo)
FOO
```

See also:

The untrace function.

trans

Translates a point (or a displacement) from one coordinate system to another

```
(trans
  pt from to [disp]
)
```

Arguments

pt A list of three reals that can be interpreted as either a 3D point or a 3D displacement (vector).

from An integer code, entity name, or 3D extrusion vector identifying the coordinate system in which *pt* is expressed. The integer code can be one of the following:

0 World (WCS)

1 User (current UCS)

2 If used with code 0 or 1, this indicates the Display Coordinate System (DCS) of the current viewport. When used with code 3, it indicates the DCS of the current model space viewport.

3 Paper space DCS (used *only* with code 2)

to An integer code, entity name, or 3D extrusion vector identifying the coordinate system of the returned point. See the *from* argument for a list of valid integer codes.

disp If present and is not `nil`, this argument specifies that *pt* is to be treated as a 3D displacement rather than as a point.

If you use an entity name for the *from* or *to* argument, it must be passed in the format returned by the **entnext**, **entlast**, **entsel**, **nentsel**, and **ssname** functions. This format lets you translate a point to and from the Object Coordinate System (OCS) of a particular object. (For some objects, the OCS is equivalent to the WCS; for these objects, conversion between OCS and WCS

is a null operation.) A 3D extrusion vector (a list of three reals) is another method of converting to and from an object's OCS. However, this does not work for those objects whose OCS is equivalent to the WCS.

Return Values

A 3D point (or displacement) in the requested *to* coordinate system.

Examples

In the following examples, the UCS is rotated 90 degrees counterclockwise around the WCS *Z* axis:

```
Command: (trans '(1.0 2.0 3.0) 0 1)
(2.0 -1.0 3.0)
Command: (trans '(1.0 2.0 3.0) 1 0)
(-2.0 1.0 3.0)
```

The coordinate systems are discussed in greater detail in Coordinate System Transformations in the *AutoLISP Developer's Guide*.

For example, to draw a line from the insertion point of a piece of text (without using Osnap), you convert the text object's insertion point from the text object's OCS to the UCS.

```
(trans
  text-insert-point
  text-ename
 1)
```

You can then pass the result to the From Point prompt.

Conversely, you must convert point (or displacement) values to their destination OCS before feeding them to **entmod**. For example, if you want to move a circle (without using the MOVE command) by the UCS-relative offset (1,2,3), you need to convert the displacement from the UCS to the circle's OCS:

```
(trans '(1 2 3) 1 circle-ename)
```

Then you add the resulting displacement to the circle's center point.

For example, if you have a point entered by the user and want to find out which end of a line it looks closer to, you convert the user's point from the UCS to the DCS.

```
(trans user-point 1 2)
```

Then you convert each of the line's endpoints from the OCS to the DCS.

```
(trans endpoint line-ename 2)
```

From there you can compute the distance between the user's point and each endpoint of the line (ignoring the *Z* coordinates) to determine which end looks closer.

The **trans** function can also transform 2D points. It does this by setting the *Z* coordinate to an appropriate value. The *Z* component used depends on the *from* coordinate system that was specified and on whether the value is to be converted as a point or as a displacement. If the value is to be converted as a displacement, the *Z* value is always 0.0; if the value is to be converted as a point, the filled-in *Z* value is determined as shown in the following table:

Converted 2D point *Z* values

From	Filled-in *Z* value
WCS	0.0
UCS	Current elevation
OCS	0.0
DCS	Projected to the current construction plane (UCS XY plane + current elevation)
PSDCS	Projected to the current construction plane (UCS XY plane + current elevation)

type

Returns the type of a specified item

```
(type
  item
)
```

Arguments

item A symbol.

Return Values

The data type of *item*. Items that evaluate to `nil` (such as unassigned symbols) return `nil`. The data type is returned as one of the atoms listed in the following table:

Data types returned by the type function

Data type	Description
ENAME	Entity names
EXRXSUBR	External ObjectARX applications
FILE	File descriptors
INT	Integers
LIST	Lists
PAGETB	Function paging table
PICKSET	Selection sets
REAL	Floating-point numbers
STR	Strings
SUBR	Internal AutoLISP functions or functions loaded from compiled FAS files Functions in LISP source files loaded from the AutoCAD Command prompt may also appear as SUBR
SYM	Symbols
USUBR	User-defined functions loaded from LISP source files

Examples

For example, given the following assignments:

```
(setq a 123 r 3.45 s "Hello!" x '(a b c))
(setq f (open "name" "r"))
```

then

```
(type 'a)
  returns
 SYM (type a)
  returns
 INT (type f)
  returns
 FILE (type r)
  returns
 REAL (type s)
  returns
 STR (type x)
  returns
 LIST (type +)
  returns
 SUBR (type nil)
  returns
 nil
```

The following code example uses the **type** function on the argument passed to it:

```
(defun isint (a)
   (if (= (type a) 'INT)
  is
TYPE
   integer?
 T
  yes, return
T nil
  no, return
nil ) )
```

U Functions

V Functions

ver

Returns a string that contains the current AutoLISP version number

```
(ver)
```

The **ver** function can be used to check the compatibility of programs.

Return Values

The string returned takes the following form:

```
"Visual LISP
  version
 (
  nn
)"
```

where *version* is the current version number and *nn* is a two-letter language description.

Examples of the two-letter language descriptions are as follows:

(de) German

(en) US/UK

(es) Spanish

(fr) French

(it) Italian

Examples

```
Command: (ver)
"Mac OS Visual LISP 2012 (en)"
```

vl-acad-defun

Defines an AutoLISP function symbol as an external subroutine

```
(vl-acad-defun
   'symbol
)
```

Arguments

symbol A symbol identifying a function.

If a function does not have the `c:` prefix, and you want to be able to invoke this function from an external ObjectARX application, you can use **vl-acad-defun** to make the function accessible.

Return Values

Unspecified.

vl-acad-undefun

Undefines an AutoLISP function symbol so it is no longer available to ObjectARX applications

```
(vl-acad-undefun
   'symbol
)
```

Arguments

symbol A symbol identifying a function.

You can use **vl-acad-undefun** to undefine a `c:` function or a function that was exposed by **vl-acad-defun**.

Return Values

`T` if successful; `nil` if unsuccessful (for example, the function was not defined in AutoLISP).

vl-bb-ref

Returns the value of a variable from the blackboard namespace

```
(vl-bb-ref
   'variable
)
```

Arguments

'*variable* A symbol identifying the variable to be retrieved.

Return Values

The value of the variable named by symbol.

Examples

Set a variable in the blackboard:

```
Command: (vl-bb-set 'foobar "Root toot toot")
"Root toot toot"
```

Use **vl-bb-ref** to retrieve the value of `foobar` from the blackboard:

```
Command: (vl-bb-ref 'foobar)
"Root toot toot"
```

See also:

The vl-bb-set (page 215) function. Sharing Data Between Namespaces in the *AutoLISP Developer's Guide* for a description of the blackboard namespace.

vl-bb-set

Sets a variable in the blackboard namespace

```
(vl-bb-set
   'symbol value
)
```

Arguments

'*symbol* A symbol naming the variable to be set.

value Any value, except a function.

Return Values

The *value* you assigned to *symbol*.

Examples

```
Command: (vl-bb-set 'foobar "Root toot toot")
"Root toot toot"
Command: (vl-bb-ref 'foobar)
"Root toot toot"
```

See also:

The vl-bb-ref (page 215) function. Sharing Data Between Namespaces in the *AutoLISP Developer's Guide* for a description of the blackboard namespace.

vl-catch-all-apply

Passes a list of arguments to a specified function and traps any exceptions

```
(vl-catch-all-apply
  '
   function list
)
```

Arguments

'function A function. The *function* argument can be either a symbol identifying a **defun**, or a **lambda** expression.

list A list containing arguments to be passed to the function.

Return Values

The result of the function call, if successful. If an error occurs, **vl-catch-all-apply** returns an error object.

Examples

If the function invoked by **vl-catch-all-apply** completes successfully, it is the same as using **apply**, as the following examples show:

```
(setq catchit (apply '/ '(50 5)))
10
```

(setq catchit (vl-catch-all-apply '/ '(50 5)))

```
10
```

The benefit of using **vl-catch-all-apply** is that it allows you to intercept errors and continue processing. See what happens when you try to divide by zero using **apply**:

(setq catchit (apply '/ '(50 0)))

```
; error: divide by zero
```

When you use **apply**, an exception occurs and an error message displays.

Here is the same operation using **vl-catch-all-apply:**

(setq catchit (vl-catch-all-apply '/ '(50 0)))

```
#<%catch-all-apply-error%>
```

The **vl-catch-all-apply** function traps the error and returns an error object. Use **vl-catch-all-error-message** to see the error message contained in the error object:

```
(vl-catch-all-error-message catchit)

"divide by zero"
```

See also:

The *error* (page 82), vl-catch-all-error-p (page 218), and vl-catch-all-error-message (page 217) functions. The Error Handling in AutoLISP topic in the *AutoLISP Developer's Guide*.

vl-catch-all-error-message

Returns a string from an error object

```
(vl-catch-all-error-message
  error-obj
)
```

Arguments

error-obj An error object returned by **vl-catch-all-apply**.

Return Values

A string containing an error message.

Examples

Divide by zero using **vl-catch-all-apply**:

```
(setq catchit (vl-catch-all-apply '/ '(50 0)))
#<%catch-all-apply-error%>
```

The **vl-catch-all-apply** function traps the error and returns an error object. Use **vl-catch-all-error-message** to see the error message contained in the error object:

```
(vl-catch-all-error-message catchit)
"divide by zero"
```

See also:

The *error* (page 82), vl-catch-all-apply (page 216), and vl-catch-all-error-p (page 218) functions. The Error Handling in AutoLISP topic in the *AutoLISP Developer's Guide*.

vl-catch-all-error-p

Determines whether an argument is an error object returned from *vl-catch-all-apply*

```
(vl-catch-all-error-p
  arg
)
```

Arguments

arg Any argument.

Return Values

`T`, if the supplied argument is an error object returned from **vl-catch-all-apply;** otherwise `nil`.

Examples

Divide by zero using **vl-catch-all-apply**:

```
(setq catchit (vl-catch-all-apply '/ '(50 0)))
```

```
#<%catch-all-apply-error%>
```

Use **vl-catch-all-error-p** to determine if the value returned by **vl-catch-all-apply** is an error object:

```
(vl-catch-all-error-p catchit)
T
```

See also:

The *error* (page 82), vl-catch-all-apply (page 216), and vl-catch-all-error-message (page 217) functions. The Error Handling in AutoLISP topic in the *AutoLISP Developer's Guide*.

vl-cmdf

Executes an AutoCAD command

Arguments

```
(vl-cmdf
   [arguments]
 ...)
```

The **vl-cmdf** function is similar to the **command** function, but differs from **command** in the way it evaluates the arguments passed to it. The **vl-cmdf** function evaluates all the supplied arguments before executing the AutoCAD command, and will not execute the AutoCAD command if it detects an error during argument evaluation. In contrast, the **command** function passes each argument in turn to AutoCAD, so the command may be partially executed before an error is detected.

If your command call includes a call to another function, **vl-cmdf** executes the call *before* it executes your command, while **command** executes the call *after* it begins executing your command.

Some AutoCAD commands may work correctly when invoked through **vl-cmdf**, while failing when invoked through **command**. The **vl-cmdf** function mainly overcomes the limitation of not being able to use **get*xxx*** functions inside command.

Arguments

arguments AutoCAD commands and their options.

The *arguments* to the **vl-cmdf** function can be strings, reals, integers, or points, as expected by the prompt sequence of the executed command. A null string ("") is equivalent to pressing Enter on the keyboard. Invoking **vl-cmdf** with no argument is equivalent to pressing Esc and cancels most AutoCAD commands.

Return Values

```
T
```

Examples

The differences between **command** and **vl-cmdf** are easier to see if you enter the following calls at the AutoCAD Command prompt, rather than the VLISP Console prompt:

```
Command: (command "line" (getpoint "point?") '(0 0) "")
line Specify first point: point?
Specify next point or [Undo]:
Command: nil
```

Using **command**, the LINE command executes first; then the **getpoint** function is called.

```
Command: (vl-cmdf "line" (getpoint "point?") '(0 0) "")
point?line Specify first point:
Specify next point or [Undo]:
Command: T
```

Using **vl-cmdf**, the **getpoint** function is called first (notice the "point?" prompt from **getpoint**); then the LINE command executes.

The following examples show the same commands, but pass an invalid point list argument to the LINE command. Notice how the results differ:

```
Command: (command "line" (getpoint "point?") '(0) "")
line Specify first point: point?
Specify next point or [Undo]:
Command: ERASE nil
Select objects: Specify opposite corner: *Cancel*
0 found
```

The **command** function passes each argument in turn to AutoCAD, without evaluating the argument, so the invalid point list is undetected.

```
Command: (vl-cmdf "line" (getpoint "point?") '(0) "")
point?Application ERROR: Invalid entity/point list.
nil
```

Because **vl-cmdf** evaluates each argument before passing the command to AutoCAD, the invalid point list is detected and the command is not executed.

See also:

The command (page 42) function.

vl-consp

Determines whether or not a list is nil

```
(vl-consp
   list-variable
)
```

The **vl-consp** function determines whether a variable contains a valid list definition.

Arguments

list-variable A list.

Return Values

`T`, if *list-variable* is a list and is not `nil`; otherwise `nil`.

Examples

```
(vl-consp nil)
nil
(vl-consp t)
nil
(vl-consp (cons 0 "LINE"))
T
```

vl-directory-files

Lists all files in a given directory

```
(vl-directory-files
   [directory pattern
```

```
    directories]
)
```

Arguments

directory A string naming the directory to collect files for; if `nil` or absent, **vl-directory-files** uses the current directory.

pattern A string containing a pattern for the file name; if `nil` or absent, **vl-directory-files** assumes "***.***"

directories An integer that indicates whether the returned list should include directory names. Specify one of the following:

-1 List directories only.

0 List files and directories (the default).

1 List files only.

Return Values

A list of file and path names; otherwise `nil` if no files match the specified pattern.

Examples

```
(vl-directory-files "/myutilities/lsp" "*.lsp")
("utilities.lsp" "blk-insert.lsp")
(vl-directory-files "/myutilities" nil -1)
("." ".." ".DS_Store" "Help" "Lsp" "Support")
```

vl-doc-ref

Retrieves the value of a variable from the current document's namespace

This function can be used by a separate-namespace VLX application to retrieve the value of a variable from the current document's namespace.

```
(vl-doc-ref
   'symbol
)
```

Arguments

'symbol A symbol naming a variable.

Return Values

The value of the variable identified by *symbol*.

Examples

```
Command: (vl-doc-ref 'foobar)
"Rinky dinky stinky"
```

See also:

The vl-doc-set (page 223) function.

vl-doc-set

Sets the value of a variable in the current document's namespace

```
(vl-doc-set
   'symbol value
)
```

This function can be used by a VLX application to set the value of a variable that resides in the current document's namespace.

If executed within a document namespace, **vl-doc-set** is equivalent to **set**.

Arguments

'symbol A symbol naming a variable.

value Any value.

Return Values

The *value* set.

Examples

```
Command: (vl-doc-set 'foobar "Rinky dinky stinky")
"Rinky dinky stinky"
```

See also:

The vl-doc-ref (page 222) function.

vl-every

Checks whether the predicate is true for every element combination

```
(vl-every
  predicate-function
   list
   [list]
...)
```

The **vl-every** function passes the first element of each supplied list as an argument to the test function, followed by the next element from each list, and so on. Evaluation stops as soon as one of the lists runs out.

Arguments

predicate-function The test function. This can be any function that accepts as many arguments as there are lists provided with **vl-every**, and returns `T` on any user-specified condition. The *predicate-function* value can take one of the following forms:

- A symbol (function name)
- `'(LAMBDA (A1 A2) ...)`
- `(FUNCTION (LAMBDA (A1 A2) ...))`

list A list to be tested.

Return Values

`T`, if *predicate-function* returns a non-`nil` value for every element combination; otherwise `nil`.

Examples

Check whether there are any empty files in the current directory:

```
(vl-every
'(lambda (fnm) (> (vl-file-size fnm) 0))
  (vl-directory-files nil nil 1) )
T
```

Check whether the list of numbers in NLST is ordered by `'<=`:

```
(setq nlst (list 0 2 pi pi 4))
```

```
(0 2 3.14159 3.14159 4)
 (vl-every '<= nlst (cdr nlst))
T
```

Compare the results of the following expressions:

```
 (vl-every '= '(1 2) '(1 3))
nil
 (vl-every '= '(1 2) '(1 2 3))
T
```

The first expression returned `nil` because **vl-every** compared the second element in each list and they were not numerically equal. The second expression returned `T` because **vl-every** stopped comparing elements after it had processed all the elements in the shorter list (1 2), at which point the lists were numerically equal. If the end of a list is reached, **vl-every** returns a non-`nil` value.

The following example demonstrates the result when **vl-every** evaluates one list that contains integer elements and another list that is `nil`:

```
 (setq alist (list 1 2 3 4))
(1 2 3 4)
 (setq junk nil)
nil
 (vl-every '= junk alist)
T
```

The return value is `T` because **vl-every** responds to the `nil` list as if it has reached the end of the list (even though the predicate hasn't yet been applied to any elements). And since the end of a list has been reached, **vl-every** returns a non-`nil` value.

vl-exit-with-error

Passes control from a VLX error handler to the *error* function of the calling namespace

```
(vl-exit-with-error
  msg
)
```

This function is used by VLX applications that run in their own namespace. When **vl-exit-with-error** executes, it calls the ***error*** function, the stack is unwound, and control returns to a command prompt.

Arguments

msg A string.

Return Values

None.

Examples

The following code illustrates the use of **vl-exit-with-error** to pass a string to the ***error*** function of the calling namespace:

```
(defun *error* (msg)
  ... ; processing in VLX namespace/execution context
(vl-exit-with-error (strcat "My application bombed! " msg)))
```

See also:

The *error* (page 82) and vl-exit-with-value (page 226) functions. The Handling Errors in an MDI Environment topic in the *AutoLISP Developer's Guide*.

vl-exit-with-value

Returns a value to the function that invoked the VLX from another namespace

```
(vl-exit-with-value
  value
)
```

A VLX ***error*** handler can use the **vl-exit-with-value** function to return a value to the program that called the VLX.

Arguments

value Any value.

Return Values

value

Examples

The following example uses **vl-exit-with-value** to return the integer value 3 to the function that invoked the VLX:

```
(defun *error* (msg)
  ... ; processing in VLX-T namespace/execution context
  (vl-exit-with-value  3))
```

See also:

The *error* (page 82) and vl-exit-with-error (page 225) functions. The Handling Errors in an MDI Environment topic in the *AutoLISP Developer's Guide*.

vl-file-copy

Copies or appends the contents of one file to another file

```
(vl-file-copy
  source-file
   destination-file
  [append]
)
```

Copy or append the contents of one file to another file. The **vl-file-copy** function will not overwrite an existing file; it will only append to it.

Arguments

source-file A string naming the file to be copied. If you do not specify a full path name, **vl-file-copy** looks in the AutoCAD default drawing directory.

destination-file A string naming the destination file. If you do not specify a path name, **vl-file-copy** writes to the AutoCAD default drawing directory.

append If specified and not `nil`, *source-file* is appended to *destination-file* (that is, copied to the end of the destination file).

Return Values

An integer, if the copy was successful; otherwise `nil`.

Some typical reasons for returning `nil` are

- *source-file* is not readable
- *source-file* is a directory
- *append?* is absent or `nil` and *destination-file* exists
- *destination-file* cannot be opened for output (that is, it is an illegal file name or a write-protected file)
- *source-file* is the same as *destination-file*

Examples

Copy *oldstart.sh* to *newstart.sh*:

```
(vl-file-copy "/oldstart.sh" "/newstart.sh")
1417
```

Copy *start.sh* to *newstart.sh*:

```
(vl-file-copy "/start.sh" "/newstart.sh")
nil
```

The copy fails because *newstart.sh* already exists, and the *append* argument was not specified.

Repeat the previous command, but specify *append*:

```
(vl-file-copy "/start.sh" "/newstart.sh" T)
185
```

The copy is successful because `T` was specified for the *append* argument.

vl-file-delete

Deletes a file

```
(vl-file-delete
  filename
)
```

Arguments

filename A string containing the name of the file to be deleted. If you do not specify a full path name, **vl-file-delete** searches the AutoCAD default drawing directory.

Return Values

T if successful; `nil` if delete failed.

Examples

Delete *newstart.sh*:

```
(vl-file-delete "newstart.sh")
nil
```

Nothing was deleted because there is no *newstart.sh* file in the AutoCAD default drawing directory.

Delete the *newstart.sh* file in the */ <root>* directory:

```
(vl-file-delete "/newstart.sh")
T
```

The delete was successful because the full path name identified an existing file.

vl-file-directory-p

Determines if a file name refers to a directory

```
(vl-file-directory-p
  filename
)
```

Arguments

filename A string containing a file name. If you do not specify a full path name, **vl-file-directory-p** searches only the AutoCAD default drawing directory.

Return Values

`T`, if *filename* is the name of a directory; `nil` if it is not.

Examples

```
(vl-file-directory-p "support")
T
(vl-file-directory-p "xyz")
nil
(vl-file-directory-p "/documents")
```

```
T
 (vl-file-directory-p "/documents/output.txt")
nil
```

vl-file-rename

Renames a file

```
(vl-file-rename
  old-filename
  new-filename
)
```

Arguments

old-filename A string containing the name of the file you want to rename. If you do not specify a full path name, **vl-file-rename** looks in the AutoCAD default drawing directory.

new-filename A string containing the new name to be assigned to the file.

NOTE If you do not specify a path name, **vl-file-rename** writes the renamed file to the AutoCAD default drawing directory.

Return Values

T, if renaming completed successfully; `nil` if renaming failed.

Examples

```
 (vl-file-rename "/oldstartup.sh" "/mystartup.sh")
T
```

NOTE

If the target file already exists, this function fails.

vl-file-size

Determines the size of a file, in bytes

```
(vl-file-size
```

```
    filename
)
```

Arguments

filename A string naming the file to be sized. If you do not specify a full path name, **vl-file-size** searches the AutoCAD default drawing directory for the file.

Return Values

If successful, **vl-file-size** returns an integer showing the size of *filename*. If the file is not readable, **vl-file-size** returns `nil`. If *filename* is a directory or an empty file, **vl-file-size** returns 0.

Examples

```
(vl-file-size "/output.txt")
1417
(vl-file-size "/")
0
```

In the preceding example, **vl-file-size** returned 0 because *c:/* names a directory.

vl-file-systime

Returns last modification time of the specified file

```
(vl-file-systime
  filename
)
```

Arguments

filename A string containing the name of the file to be checked.

Return Values

A list containing the modification date and time; otherwise `nil`, if the file is not found.

The list returned contains the following elements:

- year
- month
- day of week

- day of month
- hours
- minutes
- seconds

Note that Monday is day 1 of day of week, Tuesday is day 2, and so on.

Examples

```
(vl-file-systime "/output.txt")
(2011 5 4 26 16 3 51 586)
```

The returned value shows that the file was last modified in 2011, in the 5th month of the year (May), the 4th day of the week (Thursday), on the 26th day of the month, at 4:03:51 PM.

vl-filename-base

Returns the name of a file, after stripping out the directory path and extension

```
(vl-filename-base
  filename
)
```

Arguments

filename A string containing a file name. The **vl-filename-base** function does not check to see if the file exists.

Return Values

A string containing *filename* in uppercase, with any directory and extension stripped from the name.

Examples

```
(vl-filename-base "/myutilities/lsp/utilities.lsp")
"utilities"
(vl-filename-base "/myutilities/support")
"support"
```

vl-filename-directory

Returns the directory path of a file, after stripping out the name and extension

```
(vl-filename-directory
  filename
)
```

Arguments

filename A string containing a complete file name, including the path. The **vl-filename-directory** function does not check to see if the specified file exists.

Return Values

A string containing the directory portion of *filename*, in uppercase.

Examples

```
(vl-filename-directory "/myutilities/support/template.txt")
"/myutilities/support"
(vl-filename-directory "template.txt")
""
```

vl-filename-extension

Returns the extension from a file name, after stripping out the rest of the name

```
(vl-filename-extension
  filename
)
```

Arguments

filename A string containing a file name, including the extension. The **vl-filename-extension** function does not check to see if the specified file exists.

Return Values

A string containing the extension of *filename*. The returned string starts with a period (.) and is in uppercase. If *filename* does not contain an extension, **vl-filename-extension** returns `nil`.

Examples

```
(vl-filename-extension "/myutilities/support/output.txt")
".txt"
(vl-filename-extension "/myutilities/support/output")
nil
```

vl-filename-mktemp

Calculates a unique file name to be used for a temporary file

```
(vl-filename-mktemp
  [pattern
   directory
  extension]
)
```

Arguments

pattern A string containing a file name pattern; if `nil` or absent, **vl-filename-mktemp** uses "$VL~~".

directory A string naming the directory for temporary files; if `nil` or absent, **vl-filename-mktemp** chooses a directory in the following order:

- The directory specified in *pattern*, if any.
- The directory specified by the TEMPPREFIX system variable.
- The current directory.

extension A string naming the extension to be assigned to the file; if `nil` or absent, **vl-filename-mktemp** uses the extension part of *pattern* (which may be an empty string).

Return Values

A string containing a file name, in the following format:

```
  directory
\
  base
<
  XXX
```

```
><.
  extension
>
```

where:

base is up to 5 characters, taken from *pattern*

XXX is a 3-character unique combination

All file names generated by **vl-filename-mktemp** during a session are deleted when you exit the application.

Examples

```
(vl-filename-mktemp)
"/documents/$VL~~001"
(vl-filename-mktemp "myapp.del")
"/documents/MYAPP002.DEL"
(vl-filename-mktemp "/myutilities/temp/myapp.del")
"/myutilities/temp/MYAPP003.DEL"
(vl-filename-mktemp "/myutilities/temp/myapp.del")
"/myutilities/temp/MYAPP004.DEL"
(vl-filename-mktemp "myapp" "/myutilities/temp")
"/myutilities/temp/MYAPP005"
(vl-filename-mktemp "myapp" "/myutilities/temp" ".del")
"/myutilities/temp/MYAPP006.DEL"
```

vl-list*

Constructs and returns a list

```
(vl-list*
  object
  [object]
...)
```

Arguments

object Any LISP object.

Return Values

The **vl-list*** function is similar to **list**, but it will place the last *object* in the final **cdr** of the result list. If the last argument to **vl-list*** is an atom, the result is a dotted list. If the last argument is a list, its elements are appended to all previous arguments added to the constructed list. The possible return values from **vl-list*** are

- An atom, if a single atom *object* is specified.
- A dotted pair, if all *object* arguments are atoms.
- A dotted list, if the last argument is an atom and neither of the previous conditions is true.
- A list, if none of the previous statements is true.

Examples

(vl-list* 1)

```
1
```

(vl-list* 0 "text")

```
(0 . "TEXT")
```

(vl-list* 1 2 3)

```
(1 2 . 3)
```

(vl-list* 1 2 '(3 4))

```
(1 2 3 4)
```

See also:

The list (page 137) function.

vl-list->string

Combines the characters associated with a list of integers into a string

```
(vl-list->string
  char-codes-list
)
```

Arguments

char-codes-list A list of non-negative integers. Each integer must be less than 256.

Return Values

A string of characters, with each character based on one of the integers supplied in *char-codes-list*.

Examples

```
(vl-list->string nil)
""
(vl-list->string '(49 50))
"12"
```

See also:

The vl-string->list (page 253) function.

vl-list-length

Calculates list length of a true list

```
(vl-list-length
  list-or-cons-object
)
```

Arguments

list-or-cons-object A true or dotted list.

Return Values

An integer containing the list length if the argument is a true list; otherwise `nil` if *list-or-cons-object* is a dotted list.

Compatibility note: The **vl-list-length** function returns `nil` for a dotted list, while the corresponding Common LISP function issues an error message if the argument is a dotted list.

Examples

```
(vl-list-length nil)
0
(vl-list-length '(1 2))
2
(vl-list-length '(1 2 . 3))
nil
```

See also:

The listp (page 138) function.

vl-load-all

Loads a file into all open AutoCAD documents, and into any document subsequently opened during the current AutoCAD session

```
(vl-load-all
   filename
)
```

Arguments

filename A string naming the file to be loaded. If the file is in the AutoCAD support file search path, you can omit the path name, but you must always specify the file extension; **vl-load-all** does not assume a file type.

Return Values

Unspecified. If *filename* is not found, **vl-load-all** issues an error message.

Examples

```
(vl-load-all "/myutilities/lsp/utilities.lsp")
nil
(vl-load-all "utilities.lsp")
nil
```

vl-mkdir

Creates a directory

```
(vl-mkdir
  directoryname
)
```

Arguments

directoryname The name of the directory you want to create.

Return Values

T if successful, nil if the directory exists or if unsuccessful.

Examples

Create a directory named *mydirectory*:

```
(vl-mkdir "/mydirectory")
T
```

vl-member-if

Determines if the predicate is true for one of the list members

```
(vl-member-if
  predicate-function
  list
)
```

The **vl-member-if** function passes each element in *list* to the function specified in *predicate-function*. If *predicate-function* returns a non-nil value, **vl-member-if** returns the rest of the list in the same manner as the **member** function.

Arguments

predicate-function The test function. This can be any function that accepts a single argument and returns T for any user-specified condition. The *predicate-function* value can take one of the following forms:

- A symbol (function name)
- '(LAMBDA (A1 A2) ...)
- (FUNCTION (LAMBDA (A1 A2) ...))

list A list to be tested.

Return Values

A list, starting with the first element that passes the test and containing all elements following this in the original argument. If none of the elements passes the test condition, **vl-member-if** returns nil.

Examples

The following command draws a line:

(COMMAND "_.LINE" '(0 10) '(30 50) nil)

```
nil
```

The following command uses **vl-member-if** to return association lists describing an entity, if the entity is a line:

(vl-member-if

'(lambda (x) (= (cdr x) "AcDbLine"))

(entget (entlast)))

```
((100 . "AcDbLine") (10 0.0 10.0 0.0) (11 30.0 50.0 0.0)
(210 0.0 0.0 1.0))
```

See also:

The vl-member-if-not (page 240) function.

vl-member-if-not

Determines if the predicate is nil for one of the list members

```
(vl-member-if-not
  predicate-function
   list
)
```

The **vl-member-if-not** function passes each element in *list* to the function specified in *predicate-function*. If the function returns `nil`, **vl-member-if-not** returns the rest of the list in the same manner as the **member** function.

Arguments

predicate-function The test function. This can be any function that accepts a single argument and returns `T` for any user-specified condition. The *predicate-function* value can take one of the following forms:

- A symbol (function name)
- `'(LAMBDA (A1 A2) ...)`
- `(FUNCTION (LAMBDA (A1 A2) ...))`

list A list to be tested.

Return Values

A list, starting with the first element that fails the test and containing all elements following this in the original argument. If none of the elements fails the test condition, **vl-member-if-not** returns `nil`.

Examples

```
(vl-member-if-not 'atom '(1 "Str" (0 . "line") nil t))
((0 . "line") nil T)
```

See also:

The vl-member-if (page 239) function.

vl-position

Returns the index of the specified list item

```
(vl-position
  symbol list
)
```

Arguments

symbol Any AutoLISP symbol.

list A true list.

Return Values

An integer containing the index position of *symbol* in *list*; otherwise `nil` if *symbol* does not exist in the list.

Note that the first list element is index 0, the second element is index 1, and so on.

Examples

```
(setq stuff (list "a" "b" "c" "d" "e"))
("a" "b" "c" "d" "e")
(vl-position "c" stuff)
2
```

vl-prin1-to-string

Returns the string representation of LISP data as if it were output by the prin1 function

```
(vl-prin1-to-string
  data
)
```

Arguments

data Any AutoLISP data.

Return Values

A string containing the printed representation of *data* as if displayed by **prin1**.

Examples

```
(vl-prin1-to-string "abc")
"\"abc\""
(vl-prin1-to-string "/myutilities")
"\"/myutilities\""
(vl-prin1-to-string 'my-var)
"MY-VAR"
```

See also:

The vl-princ-to-string (page 242) function.

vl-princ-to-string

Returns the string representation of LISP data as if it were output by the princ function

```
(vl-princ-to-string
  data
)
```

Arguments

data Any AutoLISP data.

Return Values

A string containing the printed representation of *data* as if displayed by **princ**.

Examples

```
(vl-princ-to-string "abc")
"abc"
(vl-princ-to-string "/myutilities")
"/myutilities"
(vl-princ-to-string 'my-var)
"MY-VAR"
```

See also:

The vl-prin1-to-string (page 242) function.

vl-propagate

Copies the value of a variable into all open document namespaces (and sets its value in any subsequent drawings opened during the current AutoCAD session)

```
(vl-propagate
   'symbol
)
```

Arguments

symbol A symbol naming an AutoLISP variable.

Return Values

Unspecified.

Examples

```
Command: (vl-propagate 'radius)
nil
```

vl-registry-delete

Deletes the specified key or value from the Windows registry

```
(vl-registry-delete
  reg-key
    [val-name]
)
```

Arguments

reg-key A string specifying a Windows registry key.

val-name A string containing the value of the *reg-key* entry.

If *val-name* is supplied and is not `nil`, the specified value will be purged from the registry. If *val-name* is absent or `nil`, the function deletes the specified key and all of its values.

Return Values

`T` if successful; otherwise `nil`.

Examples

```
(vl-registry-write "HKEY_CURRENT_USER\\Test" "" "test data")
"test data"
(vl-registry-read "HKEY_CURRENT_USER\\Test")
"test data"
(vl-registry-delete "HKEY_CURRENT_USER\\Test")
T
```

NOTE This function cannot delete a key that has subkeys. To delete a subkey you must use **vl-registry-descendents** to enumerate all subkeys and delete all of them.

See also:

The vl-registry-descendents (page 244), vl-registry-read (page 245), and vl-registry-write (page 246) functions.

vl-registry-descendents

Returns a list of subkeys or value names for the specified registry key

```
(vl-registry-descendents
  reg-key
    [val-names]
)
```

Arguments

reg-key A string specifying a registry key.

val-names A string containing the values for the *reg-key* entry.

If *val-names* is supplied and is not `nil`, the specified value names will be listed from the registry. If *val-name* is absent or `nil`, the function displays all subkeys of *reg-key*.

Return Values

A list of strings, if successful; otherwise `nil`.

Examples

(vl-registry-descendents "HKEY_LOCAL_MACHINE\\SOFTWARE")

```
("Description" "Program Groups" "ORACLE" "ODBC" "Netscape"
"Microsoft")
```

See also:

The vl-registry-delete (page 244), vl-registry-read (page 245), and vl-registry-write (page 246) functions.

vl-registry-read

Returns data stored in the registry for the specified key/value pair

```
(vl-registry-read
  reg-key
    [val-name]
)
```

Arguments

reg-key A string specifying a registry key.

val-name A string containing the value of a registry entry.

If *val-name* is supplied and is not `nil`, the specified value will be read from the registry. If *val-name* is absent or `nil`,the function reads the specified key and all of its values.

Return Values

A string containing registry data, if successful; otherwise `nil`.

Examples

```
(vl-registry-read "HKEY_CURRENT_USER\\Test")
nil
(vl-registry-write "HKEY_CURRENT_USER\\Test" "" "test data")
"test data"
(vl-registry-read "HKEY_CURRENT_USER\\Test")
"test data"
```

See also:

The vl-registry-delete (page 244), vl-registry-descendents (page 244), and vl-registry-write (page 246) functions.

vl-registry-write

Creates a key in the registry

```
(vl-registry-write
  reg-key
   [val-name val-data]
)
```

Arguments

reg-key A string specifying a registry key.

NOTE You cannot use **vl-registry-write** for HKEY_USERS or KEY_LOCAL_MACHINE.

val-name A string containing the value of a registry entry.

val-data A string containing registry data.

If *val-name* is not supplied or is `nil`,a default value for the key is written. If *val-name* is supplied and *val-data* is not specified, an empty string is stored.

Return Values

vl-registry-write returns *val-data*, if successful; otherwise `nil`.

Examples

```
(vl-registry-write "HKEY_CURRENT_USER\\Test" "" "test data")
"test data"
(vl-registry-read "HKEY_CURRENT_USER\\Test")
"test data"
```

See also:

The vl-registry-delete (page 244), dialog box (page 244), and vl-registry-read (page 245) functions.

vl-remove

Removes elements from a list

```
(vl-remove
  element-to-remove
   list
)
```

Arguments

element-to-remove The value of the element to be removed; may be any LISP data type.

list Any list.

Return Values

The *list* with all elements except those equal to *element-to-remove*.

Examples

```
(vl-remove pi (list pi t 0 "abc"))
(T 0 "abc")
```

vl-remove-if

Returns all elements of the supplied list that fail the test function

```
(vl-remove-if
  predicate-function
   list
)
```

Arguments

predicate-function The test function. This can be any function that accepts a single argument and returns `T` for any user-specified condition. The *predicate-function* value can take one of the following forms:

- A symbol (function name)
- `'(LAMBDA (A1 A2) ...)`
- `(FUNCTION (LAMBDA (A1 A2) ...))`

list A list to be tested.

Return Values

A list containing all elements of *list* for which *predicate-function* returns `nil`.

Examples

```
(vl-remove-if 'vl-symbolp (list pi t 0 "abc"))
(3.14159 0 "abc")
```

vl-remove-if-not

Returns all elements of the supplied list that pass the test function

```
(vl-remove-if-not
  predicate-function
   list
)
```

Arguments

predicate-function The test function. This can be any function that accepts a single argument and returns `T` for any user-specified condition. The *predicate-function* value can take one of the following forms:

- A symbol (function name)
- `'(LAMBDA (A1 A2) ...)`

- `(FUNCTION (LAMBDA (A1 A2) ...))`

list A list to be tested.

Return Values

A list containing all elements of *list* for which *predicate-function* returns a non-`nil` value

Examples

```
(vl-remove-if-not 'vl-symbolp (list pi t 0 "abc"))
(T)
```

vl-some

Checks whether the predicate is not nil for one element combination

```
(vl-some
  predicate-function
  list
   [list]
...)
```

Arguments

predicate-function The test function. This can be any function that accepts as many arguments as there are lists provided with **vl-some**, and returns `T` on a user-specified condition. The *predicate-function* value can take one of the following forms:

- A symbol (function name)
- `'(LAMBDA (A1 A2) ...)`
- `(FUNCTION (LAMBDA (A1 A2) ...))`

list A list to be tested.

The **vl-some** function passes the first element of each supplied list as an argument to the test function, then the next element from each list, and so on. Evaluation stops as soon as the predicate function returns a non-`nil` value for an argument combination, or until all elements have been processed in one of the lists.

Return Values

The predicate value, if *predicate-function* returned a value other than `nil`; otherwise `nil`.

Examples

The following example checks whether `nlst` (a number list) has equal elements in sequence:

```
(setq nlst (list 0 2 pi pi 4))
(0 2 3.14159 3.14159 4)
(vl-some '= nlst (cdr nlst))
T
```

vl-sort

Sorts the elements in a list according to a given compare function

```
(vl-sort
  list
  comparison-function
)
```

Arguments

list Any list.

comparison-function A comparison function. This can be any function that accepts two arguments and returns T (or any non-`nil` value) if the first argument precedes the second in the sort order. The *comparison-function* value can take one of the following forms:

- A symbol (function name)
- `'(LAMBDA (A1 A2) ...)`
- `(FUNCTION (LAMBDA (A1 A2) ...))`

Return Values

A list containing the elements of *list* in the order specified by *comparison-function*. Duplicate elements may be eliminated from the list.

Examples

Sort a list of numbers:

```
  (vl-sort '(3 2 1 3) '<)
(1 2 3) ;
```

Note that the result list contains only one 3.

Sort a list of 2D points by Y coordinate:

```
_$
  (vl-sort '((1 3) (2 2) (3 1))
           (function (lambda (e1 e2)
                     (< (cadr e1) (cadr e2)) ) ) )
((3 1) (2 2) (1 3))
```

Sort a list of symbols:

```
_$
  (vl-sort
    '(a d c b a)
    '(lambda (s1 s2)
     (< (vl-symbol-name s1) (vl-symbol-name s2)) ) )
 (A B C D) ; Note that only one A remains in the result
list
```

vl-sort-i

Sorts the elements in a list according to a given compare function, and returns the element index numbers

```
(vl-sort-i  list
  comparison-function
)
```

Arguments

list Any list.

comparison-function A comparison function. This can be any function that accepts two arguments and returns T (or any non-`nil` value) if the first

argument precedes the second in the sort order. The *comparison-function* value can take one of the following forms:

- A symbol (function name)
- `'(LAMBDA (A1 A2) ...)`
- `(FUNCTION (LAMBDA (A1 A2) ...))`

Return Values

A list containing the index values of the elements of *list*, sorted in the order specified by *comparison-function*. Duplicate elements will be retained in the result.

Examples

Sort a list of characters in descending order:

```
(vl-sort-i '("a" "d" "f" "c") '>)
(2 1 3 0)
```

The sorted list order is “f” “d” “c” “a”; “f” is the 3rd element (index 2) in the original list, “d” is the 2nd element (index 1) in the list, and so on.

Sort a list of numbers in ascending order:

```
(vl-sort-i '(3 2 1 3) '<)
(2 1 3 0)
```

Note that both occurrences of 3 are accounted for in the result list.

Sort a list of 2D points by *Y* coordinate:

```
(vl-sort-i '((1 3) (2 2) (3 1))
      (function (lambda (e1 e2)
          (< (cadr e1) (cadr e2)) ) ) )
(2 1 0)
```

Sort a list of symbols:

```
(vl-sort-i
  '(a d c b a)
  '(lambda (s1 s2)
   (< (vl-symbol-name s1) (vl-symbol-name s2)) ) )
(4 0 3 2 1)
```

Note that both a's are accounted for in the result list.

vl-string->list

Converts a string into a list of character codes

```
(vl-string->list
  string
)
```

Arguments

string A string.

Return Values

A list, each element of which is an integer representing the character code of the corresponding character in *string*.

Examples

```
(vl-string->list "")
nil
(vl-string->list "12")
(49 50)
```

See also:

The vl-list->string (page 236) function.

vl-string-elt

Returns the ASCII representation of the character at a specified position in a string

```
(vl-string-elt
  string position
)
```

Arguments

string A string to be inspected.

position A displacement in the string; the first character is displacement 0. Note that an error occurs if *position* is outside the range of the string.

Return Values

An integer denoting the ASCII representation of the character at the specified position.

Examples

```
(vl-string-elt "May the Force be with you" 8)
70
```

vl-string-left-trim

Removes the specified characters from the beginning of a string

```
(vl-string-left-trim
  character-set
   string
)
```

Arguments

character-set A string listing the characters to be removed.

string The string to be stripped of *character-set*.

Return Values

A string containing a substring of *string* with all leading characters in *character-set* removed

Examples

```
(vl-string-left-trim " \t\n" "\n\t STR ")
"STR "
(vl-string-left-trim "12456789" "12463CPO is not R2D2")
"3CPO is not R2D2"
(vl-string-left-trim " " "    There are too many spaces here")
"There are too many spaces here"
```

vl-string-mismatch

Returns the length of the longest common prefix for two strings, starting at specified positions

```
(vl-string-mismatch
  str1
  str2
   [pos1
  pos2
  ignore-case-p]
)
```

Arguments

str1 The first string to be matched.

str2 The second string to be matched.

pos1 An integer identifying the position to search from in the first string; 0 if omitted.

pos2 An integer identifying the position to search from in the second string; 0 if omitted.

ignore-case-**p** If `T` is specified for this argument, case is ignored; otherwise, case is considered.

Return Values

An integer.

Examples

```
(vl-string-mismatch "VL-FUN" "VL-VAR")
3
(vl-string-mismatch "vl-fun" "avl-var")
0
(vl-string-mismatch "vl-fun" "avl-var" 0 1)
3
(vl-string-mismatch "VL-FUN" "Vl-vAR")
1
(vl-string-mismatch "VL-FUN" "Vl-vAR" 0 0 T)
3
```

vl-string-position

Looks for a character with the specified ASCII code in a string

```
(vl-string-position
  char-code
  str
  [start-pos
   [from-end-p]]
)
```

Arguments

char-code The integer representation of the character to be searched.

str The string to be searched.

start-pos The position to begin searching from in the string (first character is 0); 0 if omitted.

from-end-p If `T` is specified for this argument, the search begins at the end of the string and continues backward to *pos*.

Return Values

An integer representing the displacement at which *char-code* was found from the beginning of the string; `nil` if the character was not found.

Examples

```
(vl-string-position (ascii "z") "azbdc")
1
(vl-string-position 122 "azbzc")
1
(vl-string-position (ascii "x") "azbzc")
nil
```

The search string used in the following example contains two "z" characters. Reading from left to right, with the first character being displacement 0, there is one z at displacement 1 and another z at displacement 3:

```
(vl-string-position (ascii "z") "azbzlmnqc")
1
```

Searching from left to right (the default), the "z" in position 1 is the first one **vl-string-position** encounters. But when searching from right to left, as in the following example, the "z" in position 3 is the first one encountered:

```
(vl-string-position (ascii "z") "azbzlmnqc" nil t)
3
```

vl-string-right-trim

Removes the specified characters from the end of a string

```
(vl-string-right-trim
  character-set
   string
)
```

Arguments

character-set A string listing the characters to be removed.

string The string to be stripped of *character-set*.

Return Values

A string containing a substring of *string* with all trailing characters in *character-set* removed.

Examples

```
(vl-string-right-trim " \t\n" " STR \n\t ")
" STR"
(vl-string-right-trim "1356789" "3CPO is not R2D267891")
"3CPO is not R2D2"
(vl-string-right-trim " " "There are too many spaces here      ")
"There are too many spaces here"
```

vl-string-search

Searches for the specified pattern in a string

```
(vl-string-search
```

```
    pattern
     string
     [start-pos]
)
```

Arguments

pattern A string containing the pattern to be searched for.

string The string to be searched for *pattern*.

start-pos An integer identifying the starting position of the search; 0 if omitted.

Return Values

An integer representing the position in the string where the specified *pattern* was found; otherwise `nil` if the pattern is not found; the first character of the string is position 0.

Examples

```
(vl-string-search "foo" "pfooyey on you")
1
(vl-string-search "who" "pfooyey on you")
nil
(vl-string-search "foo" "fooey-more-fooey" 1)
11
```

vl-string-subst

Substitutes one string for another, within a string

```
(vl-string-subst
  new-str
  pattern
   string
   [start-pos]
)
```

Arguments

new-str The string to be substituted for *pattern*.

pattern A string containing the pattern to be replaced.

string The string to be searched for *pattern*.

start-pos An integer identifying the starting position of the search; 0 if omitted.

Note that the search is case-sensitive, and that **vl-string-subst** substitutes only the first occurrence it finds of the string.

Return Values

The value of *string* after any substitutions have been made.

Examples

Replace the string "Ben" with "Obi-wan":

```
(vl-string-subst "Obi-wan" "Ben" "Ben Kenobi")
"Obi-wan Kenobi"
```

Replace "Ben" with "Obi-wan":

```
(vl-string-subst "Obi-wan" "Ben" "ben Kenobi")
"ben Kenobi"
```

Nothing was substituted because **vl-string-subst** did not find a match for "Ben"; the "ben" in the string that was searched begins with a lowercase "b".

Replace "Ben" with "Obi-wan":

```
(vl-string-subst "Obi-wan" "Ben" "Ben Kenobi Ben")
"Obi-wan Kenobi Ben"
```

Note that there are two occurrences of "Ben" in the string that was searched, but **vl-string-subst** replaces only the first occurrence.

Replace "Ben" with "Obi-wan," but start the search at the fourth character in the string:

```
(vl-string-subst "Obi-wan" "Ben" "Ben \"Ben\" Kenobi" 3)
"Ben \"Obi-wan\" Kenobi"
```

There are two occurrences of "Ben" in the string that was searched, but because **vl-string-subst** was instructed to begin searching at the fourth character, it found and replaced the second occurrence, not the first.

vl-string-translate

Replaces characters in a string with a specified set of characters

```
(vl-string-translate
  source-set
  dest-set
  str
)
```

Arguments

source-set A string of characters to be matched.

dest-set A string of characters to be substituted for those in *source-set*.

str A string to be searched and translated.

Return Values

The value of *str* after any substitutions have been made

Examples

```
(vl-string-translate "abcABC" "123123" "A is a, B is b, C is C")
"1 is 1, 2 is 2, 3 is 3"
(vl-string-translate "abc" "123" "A is a, B is b, C is C")
"A is 1, B is 2, C is 3"
```

vl-string-trim

Removes the specified characters from the beginning and end of a string

```
(vl-string-trim
  char-set
   str
)
```

Arguments

char-set A string listing the characters to be removed.

str The string to be trimmed of *char-set*.

Return Values

The value of *str*, after any characters have been trimmed.

Examples

```
(vl-string-trim " \t\n" " \t\n STR \n\t ")
"STR"
(vl-string-trim "this is junk" "this is junk Don't call this junk!
 this is junk")
"Don't call this junk!"
(vl-string-trim " " "     Leave me alone   ")
"Leave me alone"
```

vl-symbol-name

Returns a string containing the name of a symbol

```
(vl-symbol-name
  symbol
)
```

Arguments

symbol Any LISP symbol.

Return Values

A string containing the name of the supplied symbol argument, in uppercase.

Examples

```
(vl-symbol-name 'S::STARTUP)
"S::STARTUP"
(progn (setq sym 'my-var) (vl-symbol-name sym))
"MY-VAR"
(vl-symbol-name 1)
; *** ERROR: bad argument type: symbolp 1
```

vl-symbol-value

Returns the current value bound to a symbol

```
(vl-symbol-value
   symbol
)
```

This function is equivalent to the **eval** function, but does not call the LISP evaluator.

Arguments

symbol Any LISP symbol.

Return Values

The value of *symbol*, after evaluation.

Examples

```
(vl-symbol-value 't)
T
(vl-symbol-value 'PI)
3.14159
(progn (setq sym 'PAUSE) (vl-symbol-value sym))
"\\"
```

vl-symbolp

Identifies whether or not a specified object is a symbol

Arguments

```
(vl-symbolp
  object
)
```

object Any LISP object.

Return Values

`T` if *object* is a symbol; otherwise `nil`.

Examples

```
(vl-symbolp t)
T
(vl-symbolp nil)
nil
(vl-symbolp 1)
nil
(vl-symbolp (list 1))
nil
```

vports

Returns a list of viewport descriptors for the current viewport configuration

```
(vports)
```

Return Values

One or more viewport descriptor lists consisting of the viewport identification number and the coordinates of the viewport's lower-left and upper-right corners.

If the AutoCAD TILEMODE system variable is set to 1 (on), the returned list describes the viewport configuration created with the AutoCAD VPORTS command. The corners of the viewports are expressed in values between 0.0 and 1.0, with (0.0, 0.0) representing the lower-left corner of the display screen's graphics area, and (1.0, 1.0) the upper-right corner. If TILEMODE is 0 (off), the returned list describes the viewport objects created with the MVIEWcommand. The viewport object corners are expressed in paper space coordinates. Viewport number 1 is always paper space when TILEMODE is off.

Examples

Given a single-viewport configuration with TILEMODE on, the **vports** function might return the following:

```
((1 (0.0 0.0) (1.0 1.0)))
```

Given four equal-sized viewports located in the four corners of the screen when TILEMODE is on, the **vports** function might return the following lists:

```
((5 (0.5 0.0) (1.0 0.5))
 (2 (0.5 0.5) (1.0 1.0))
 (3 (0.0 0.5) (0.5 1.0))
 (4 (0.0 0.0) (0.5 0.5)) )
```

The current viewport's descriptor is always first in the list. In the previous example, viewport number 5 is the current viewport.

W Functions

wcmatch

Performs a wild-card pattern match on a string

```
(wcmatch
   string pattern
)
```

Arguments

string A string to be compared. The comparison is case-sensitive, so uppercase and lowercase characters must match.

pattern A string containing the pattern to match against *string*. The *pattern* can contain the wild-card pattern-matching characters shown in the table Wild-card characters. You can use commas in a pattern to enter more than one pattern condition. Only the first 500 characters (approximately) of the *string* and *pattern* are compared; anything beyond that is ignored.

Both arguments can be either a quoted string or a string variable. It is valid to use variables and values returned from AutoLISP functions for *string* and *pattern* values.

Return Values

If *string* and *pattern* match, **wcmatch** returns `T`; `otherwise`, **wcmatch** returns `nil`.

Wild-card characters

Character	Definition
`#` (pound)	Matches any single numeric digit.
`@` (at)	Matches any single alphabetic character.
`.` (period)	Matches any single nonalphanumeric character.
`*` (asterisk)	Matches any character sequence, including an empty one, and it can be used anywhere in the search pattern: at the beginning, middle, or end.
`?` (question mark)	Matches any single character.
`~` (tilde)	If it is the first character in the pattern, it matches anything except the pattern.
`[...]`	Matches any one of the characters enclosed.
`[~...]`	Matches any single character not enclosed.
`-` (hyphen)	Used inside brackets to specify a range for a single character.
`,` (comma)	Separates two patterns.
`` ` `` (reverse quote)	Escapes special characters (reads next character literally).

Examples

The following command tests a string to see if it begins with the character `N`:

```
Command: (wcmatch "Name" "N*")
T
```

The following example performs three comparisons. If any of the three pattern conditions is met, **wcmatch** returns T. The tests are:

- Does the string contain three characters?
- Does the string not contain an m?
- Does the string begin with the letter "N"?

If any of the three pattern conditions is met, **wcmatch** returns T:

```
Command: (wcmatch "Name" "???,~*m*,N*")
T
```

In this example, the last condition was met, so **wcmatch** returned T.

Using Escape Characters with wcmatch

To test for a wild-card character in a string, you can use the single reverse-quote character (`) to *escape* the character. *Escape* means that the character following the single reverse quote is not read as a wild-card character; it is compared at its face value. For example, to search for a comma anywhere in the string "Name", enter the following:

```
Command: (wcmatch "Name" "*`,*")
nil
```

Both the C and AutoLISP programming languages use the backslash (\) as an escape character, so you need two backslashes (\\) to produce one backslash in a string. To test for a backslash character anywhere in "Name", use the following function call:

```
Command: (wcmatch "Name" "*`\\*")
nil
```

All characters enclosed in brackets ([. . .]) are read literally, so there is no need to escape them, with the following exceptions: the tilde character (~) is read literally only when it is not the first bracketed character (as in "[A~BC]"); otherwise, it is read as the negation character, meaning that **wcmatch** should match all characters except those following the tilde (as in "[~ABC]"). The dash character (-) is read literally only when it is the first or last bracketed character (as in "[-ABC]" or "[ABC-]") or when it follows a leading tilde (as in "[~-ABC]"). Otherwise, the dash character (-) is used within brackets to specify a range of values for a specific character. The range works only for single characters, so "STR[1-38]" matches STR1, STR2, STR3, and STR8, and "[A-Z]" matches any single uppercase letter.

The closing bracket character (]) is also read literally if it is the first bracketed character or if it follows a leading tilde (as in "[ ]ABC]" or "[~]ABC]").

NOTE

Because additional wild-card characters might be added in future releases of AutoLISP, it is a good idea to escape all nonalphanumeric characters in your pattern to ensure upward compatibility.

while

Evaluates a test expression, and if it is not nil, evaluates other expressions; repeats this process until the test expression evaluates to nil

```
(while
  testexpr [expr
...
  ]
)
```

The **while** function continues until *testexpr* is `nil`.

Arguments

testexpr The expression containing the test condition.

expr One or more expressions to be evaluated until *testexpr* is `nil`.

Return Values

The most recent value of the last *expr*.

Examples

The following code calls user function **some-func** ten times, with `test` set to 1 through 10. It then returns 11, which is the value of the last expression evaluated:

```
(setq test 1)
(while (<= test 10)
  (some-func test)
  (setq test (1+ test))
)
```

write-char

Writes one character to the screen or to an open file

```
(write-char
  num [file-desc]
)
```

Arguments

num The decimal ASCII code for the character to be written.

file-desc A file descriptor for an open file.

Return Values

The *num* argument.

Examples

The following command writes the letter *C* to the command window, and returns the supplied *num* argument:

```
Command: (write-char 67)
C67
```

Assuming that `f` is the descriptor for an open file, the following command writes the letter *C* to that file:

```
Command: (write-char 67 f)
67
```

Note that **write-char** cannot write a NULL character (ASCII code 0) to a file.

write-line

Writes a string to the screen or to an open file

```
(write-line
  string [file-desc]
)
```

Arguments

string A string.

file-desc A file descriptor for an open file.

Return Values

The *string*, quoted in the normal manner. The quotes are omitted when writing to a file.

Examples

Open a new file:

```
Command: (setq f (open "/documents/new.txt" "w"))
#<file "/documents/new.txt">
```

Use **write-line** to write a line to the file:

```
Command: (write-line "To boldly go where nomad has gone before." f)
"To boldly go where nomad has gone before."
```

The line is not physically written until you close the file:

```
Command: (close f)
nil
```

X Functions

xdroom

Returns the amount of extended data (xdata) space that is available for an object (entity)

```
(xdroom
  ename
)
```

Because there is a limit (currently, 16 kilobytes) on the amount of extended data that can be assigned to an entity definition, and because multiple applications can append extended data to the same entity, this function is provided so an application can verify there is room for the extended data that it will append. It can be called in conjunction with **xdsize**, which returns the size of an extended data list.

Arguments

ename An entity name (ename data type).

Return Values

An integer reflecting the number of bytes of available space. If unsuccessful, **xdroom** returns `nil`.

Examples

The following example looks up the available space for extended data of a viewport object:

```
Command: (xdroom vpname)
16162
```

In this example, 16,162 bytes of the original 16,383 bytes of extended data space are available, meaning that 221 bytes are used.

xdsize

Returns the size (in bytes) that a list occupies when it is linked to an object (entity) as extended data

```
(xdsize
  lst
)
```

Arguments

lst A valid list of extended data that contains an application name previously registered with the use of the **regapp** function. See the Examples (page 270) section of this function for *lst* examples.

Return Values

An integer reflecting the size, in bytes. If unsuccessful, **xdsize** returns `nil`.

Brace fields (group code 1002) must be balanced. An invalid *lst* generates an error and places the appropriate error code in the ERRNO variable. If the extended data contains an unregistered application name, you see this error message (assuming that CMDECHO is on):

```
Invalid application name in 1001 group
```

Examples

The *lst* can start with a -3 group code (the extended data sentinel), but it is not required. Because extended data can contain information from multiple applications, the list must have a set of enclosing parentheses.

```
(-3 ("MYAPP" (1000 . "SUITOFARMOR")
```

```
            (1002 . "{")
            (1040 . 0.0)
            (1040 . 1.0)
            (1002 . "}")
    )
  )
```

Here is the same example without the -3 group code. This list is just the **cdr** of the first example, but it is important that the enclosing parentheses are included:

```
  ( ("MYAPP" (1000 . "SUITOFARMOR")
             (1002 . "{")
             (1040 . 0.0)
             (1040 . 1.0)
             (1002 . "}")
    )
  )
```

Z Functions

zerop

Verifies that a number evaluates to zero

```
(zerop
  number
)
```

Arguments

number A number.

Return Values

`T` if *number* evaluates to zero; otherwise `nil`.

Examples

```
Command: (zerop 0)
T
Command: (zerop 0.0)
T
```

```
Command: (zerop 0.0001)
nil
```

Externally Defined Commands

2

Externally Defined Commands

AutoCAD® commands defined by ObjectARX® or AutoLISP® applications are called externally defined. AutoLISP applications may need to access externally defined commands differently from the way they access built-in AutoLISP functions. Many externally defined commands have their own programming interfaces that allow AutoLISP applications to take advantage of their functionality.

For additional information on the commands described in this appendix, see the *Command Reference*.

align

Translates and rotates objects, allowing them to be aligned with other objects.

NOTE The Geom3d ObjectARX application must be loaded before the function can be called, `(arxload "geom3d")`.

```
(align
  arg1 arg2 ...
)
```

Arguments

arg1 arg2... Arguments to the AutoCAD ALIGN command. The order, number, and type of arguments for the **align** function are the same as if you were entering ALIGN at the command line.

To indicate a null response (a user pressing Enter), specify `nil` or an empty string (`""`).

Return Values

`T` if successful; otherwise `nil`.

Examples

The following example specifies two pairs of source and destination points, which perform a 2D move:

```
(setq ss (ssget))
(align ss s1 d1 s2 d2 "" "2d")
```

cal

Invokes the on-line geometry calculator and returns the value of the evaluated expression (externally defined: geomcal ObjectARX application)

```
(c:cal
  expression
)
```

Arguments

expression A quoted string. See CAL in the *Command Reference* for a description of allowable expressions.

Return Values

The result of the expression.

Examples

The following example uses **cal** in an AutoLISP expression with the **trans** function:

```
(trans (c:cal "[1,2,3]+MID") 1 2)
```

mirror3d

Reflects selected objects about a user-specified plane.

NOTE The Geom3d ObjectARX application must be loaded before the function can be called, `(arxload "geom3d")`.

```
(mirror3d
  arg1 arg2 ...
)
```

Arguments

The order, number, and type of arguments for the **mirror3d** function are the same as if you were entering the MIRROR3D AutoCAD command. To signify a user pressing Enter without typing any values, use `nil` or an empty string (`""`).

Return Values

`T` if successfu;, otherwise `nil`.

Examples

The following example mirrors the selected objects about the *XY* plane that passes through the point 0,0,5, and then deletes the old objects:

```
(setq ss (ssget))
(mirror3d ss "XY" '(0 0 5) "Y")
```

rotate3d

Rotates an object about an arbitrary 3D axis.

NOTE The Geom3d ObjectARX application must be loaded before the function can be called, `(arxload "geom3d")`.

```
(rotate3d
 args ...
)
```

Arguments

args The order, number, and type of arguments for the **rotate3d** function are the same as if you were entering them at the command line; see ROTATE3D in the *Command Reference*.

To signify a null response (user pressing Enter without specifying any arguments), use `nil` or an empty string (`""`).

Return Values

If successful, **rotate3d** returns `T`; otherwise it returns `nil`.

Examples

The following example rotates the selected objects 30 degrees about the axis specified by points *p1* and *p2*.

```
(setq ss (ssget))
(rotate3d ss p1 p2 30)
```

AutoLISP support for the **rotate3d** function is implemented with the use of the SAGET library.

solprof

Creates profile images of three-dimensional solids.

NOTE The AcSolids ObjectARX application must be loaded before the function can be called, `(arxload "acsolids")`.

```
(c:solprof
  args ...
)
```

Arguments

args The order, number, and type of arguments are the same as those specified when issuing SOLPROF at the Command prompt.

Index

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y

Z